DIGITAL WATERMARKING FOR COMPACT DISCS AND THEIR EFFECT ON THE ERROR CORRECTION SYSTEM

by

KAY RYDYGER

A thesis submitted to the University of Plymouth
in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

Centre for Research in Information Storage Technology
Department of Communication and Electronic Engineering
Faculty of Technology

February 2002

90 0508783 1

UNIVERSITY OF PLYMOUTH
Item No. 9005087831
Date 30 MAY 2002 T
Class No. THESIS 005.8 RYD
Cont. No. X704426 42X
PLYMOUTH LIBRARY

REFERENCE ONLY

LIBRARY STORE

Digital Watermarking for Compact Discs and their Effect on the Error Correction System

by

Kay Rydyger

A new technique, based on current compact disc technology, to image the transparent surface of a compact disc, or additionally the reflective information layer, has been designed, implemented and evaluated. This technique (image capture technique) has been tested and successfully applied to the detection of mechanically introduced compact disc watermarks and biometrical information with a resolution of $1.6\mu m \times 14\mu m$.

Software has been written which, when used with the image capture technique, recognises a compact disc based on its error distribution. The software detects digital watermarks which cause either laser signal distortions or decoding error events. Watermarks serve as secure media identifiers.

The complete channel coding of a Compact Disc Audio system including EFM modulation, error-correction and interleaving have been implemented in software. The performance of the error correction system of the compact disc has been assessed using this simulation model. An embedded data channel holding watermark data has been investigated. The covert channel is implemented by means of the error-correction ability of the Compact Disc system and was realised by aforementioned techniques like engraving the reflective layer or the polysubstrate layer. Computer simulations show that watermarking schemes, composed of regularly distributed single errors, impose a minimum effect on the error correction system.

Error rates increase by a factor of ten if regular single-symbol errors per frame are introduced - all other patterns further increase the overall error rates. Results show that background signal noise has to be reduced by a factor of 60% to account for the additional burden of this optimal watermark pattern.

Two decoding strategies, usually employed in modern CD decoders, have been examined. Simulations take emulated bursty background noise as it appears in user-handled discs into account. Variations in output error rates, depending on the decoder and the type of background noise became apparant. At low error rates ($r < 0.003$) the output symbol error rate for a bursty background differs by 20% depending on the decoder. Differences between a typical burst error distribution caused by user-handling

and a non-burst error distribution has been found to be approximately 1% with the higher performing decoder.

Simulation results show that the drop of the error-correction rates due to the presence of a watermark pattern quantitatively depends on the characteristic type of the background noise. A four times smaller change to the overall error rate was observed when adding a regular watermark pattern to a characteristic background noise, as caused by user-handling, compared to a non-bursty background.

Contents

1 Introduction **1**

1.1 The Compact Disc 1

1.2 The Compact Disc's Vulnerability to Errors 4

1.2.1 Coping with Errors on Compact Discs 4

1.2.2 Manufacturing Errors on Compact Discs 5

1.2.3 Blemishes resulting from User-handling 6

1.2.4 Definitions 7

1.3 Compact Disc Standards 8

1.4 Contribution to Knowledge 10

1.5 Organisation of the Thesis 11

2 Background to the Investigation **12**

2.1 The Compact Disc System - Benefits and Problems 12

2.2 Watermarking and Steganography on Compact Discs 13

2.3 Scanning Microscopy 18

2.4 Error Correction Simulations 19

2.5 Measuring Error Distributions 22

2.6 Performance of Compact Disc Player's Decoder Strategies 22

3 Experimental Apparatus and Techniques **26**

3.1 A CD-based Image Capture System 26

3.1.1 Introduction 26

3.1.2 Basic Construction of a Compact Disc Player 27

3.1.3 Presentation of the Novel Capture System 28

3.1.4 Necessary Hardware Alterations to the Standard CD-Device . . 31

3.1.5 Control and Data Acquisition Software of the Capture System . 33
3.1.6 Capabilities of Capture System 37
3.2 Modelling the Error Recovery Process 38
3.2.1 Experimental Apparatus versus Simulation Techniques 38
3.2.2 Basics about the Channel Coding for Compact Discs 39
3.3 Software Implementation of the Channel Coding and Decoding 42
3.3.1 Introduction 42
3.3.2 Implementation of EFM-Modulation 43
3.3.3 Implementation of the Interleaver 44
3.3.4 Cross Interleave Reed-Solomon Decoder Schemes 44
3.3.5 Sub-code Channel and Synchronisation Patterns 45
3.3.6 Random Number Generator 46
3.3.7 Control of the Software 46
3.3.8 Computing Environment and Tests 48

4 Results of Graphical Capture System 50
4.1 Methodology 50
4.2 Properties of the System 52
4.2.1 Resolution of the Scanning Process 52
4.2.2 Signal-to-Noise Ratio of the Read-Out Signal 54
4.2.3 Using Different Compact Disc Players 55
4.3 Data and Image Processing Software 57
4.4 Interpretation of Results 60
4.4.1 Detecting Handwriting on the CD Surface 60
4.4.2 Fingerprints on Compact Discs 61
4.4.3 Watermarks and their Capturing on Compact Discs 63
4.5 Further Applications 69
4.5.1 Random User-Handling and Dirt 69
4.5.2 Moiré-Pattern 70
4.5.3 Detecting Manufacturing Defects during Read-out 72
4.6 Conclusions and Applicability for Watermarking 77

5 Watermarking and Punctured Code Simulations **79**

5.1 Definitions and Background 79

5.1.1 Error Concealment 79

5.1.2 Definitions of Error Rates 81

5.1.3 Limits of Computer Simulation 81

5.2 Conformance of Computer Simulation and Statistical Analysis 83

5.3 Decoder Algorithms applied to Memoryless Channels 83

5.4 Decoding Burst Errors 86

5.4.1 Introduction 86

5.4.2 Reproduction of Error Burst Probabilities and Good Data Gap Probabilities in a Bursty Channel 87

5.4.3 Decoder Algorithms Applied to Bursty Channels 90

5.4.4 Error Correction Capacity on a Bursty Channel and Memoryless Channel 93

5.4.5 Discussion 93

5.5 Performance of Watermark Sequences 96

5.5.1 Introduction 96

5.5.2 Motivation for Considering Background Noise 98

5.5.3 Using Intentional Errors as Watermarks 100

5.5.4 Choosing non EFM-Words as Error Symbols 114

5.5.5 Changes to the Overall Error Rate when Introducing Watermarks 117

5.5.6 Conclusions 119

6 Conclusions and Discussions **122**

6.1 Conclusions 122

6.2 Future Work 132

A Theory of Reed-Solomon Encoding and Decoding for Compact Discs 134

A.1 Galois-field Arithmetic and Basics of ECC 134

A.2 Cyclic Code Encoding 136

A.3 Reed-Solomon Syndrome Decoding 137

B Software Listings **140**

B.1 Control Software for the Experimental Apparatus 140

B.2 Simulation Software for Compact Disc Channel Modelling 149
B.2.1 Flowchart of the Encoder and Decoder Program 149
B.2.2 Program listing 149

List of Abbreviations

ADC	Analog-to-Digital Converter
CD	Compact Disc
CD-DA	Compact Disc Digital Audio
CD-R	Compact Disc Recordable
CD-ROM	Compact Disc Read Only Memory
CD-RW	Compact Disc Rewritable
CIRC	Cross-Interleave-Reed-Solomon-Code
CLV	constant linear velocity
codec	Coder/Decoder
CRC	Cyclic Redundancy Check
DRM	Digital Rights Management
DVD	Digital Versatile Disc
ECC	Error Control Code
EC	Error correction
EFM	Eight-To-Fourteen Modulation
GF	Galois Field
HP	Hewlett Packard
LSB	least significant bit
NRZI	non-return to zero inverted
NRZ	non-return to zero
PCB	Printed Circuit Board
PC	Personal Computer

PDF	probability density function
PPM	Portable Pixmap file format
RF	Radio Frequency
RLL	run-length-limited
rpm	rounds per minute
SCSI	Small Computer System Interface
SNR	Signal to Noise Ratio

List of Figures

1.1 Schematic view of defect classification in a Compact Disc. 6

3.1 Structure of the experimental setup 27
3.2 Servo circuits in a compact disc player 28
3.3 Structure of the image capture system. The photo shows the opened compact disc player and the trigger circuitry. 31
3.4 Schematic diagram of the interface trigger to ADC. 33
3.5 Bitstream of the encoding system of the Compact Disc. 40
3.6 CIRC decoder structure 42

4.1 Photograph of radial scratches on compact disc. 52
4.2 Laser-signal response to five radial scratches in compact disc. 53
4.3 Voltage drop due to a black stripe on surface. The other peaks signify surface scratches. 54
4.4 Signal response of two different players to scratches on the compact disc. a) Toshiba XM3301B, b) Toshiba XM4101B 55
4.5 Fourier transform of data in Figure 4.4a (XM3301B). 56
4.6 Fourier transform of data in Figure 4.4b (XM4101B). 56
4.7 Captured overview of a disc surface with random scratches. 59
4.8 Image capture of the words "KAY" and "CRIST" written on the surface of a compact disc with a ball-point pen. 60
4.9 Captured fingerprint on the surface of a disc. The image was processed using an emboss filter. 61
4.10 A second captured fingerprint with different filter operations applied (contrast stretching). 62

4.11 Captured surface image of three punctured distortions, compact disc plays without errors. The true size of the distortions is about ten times smaller. 65
4.12 Four punctured distortions in the reflective layer, imaged by the author's capturing technique. The size of the small dot is about $100\mu m$. 66
4.13 Photograph of this distortions taken by an optical microscope. Compare to Figure 4.12. 67
4.14 Photograph of the same four distortions taken by a scanning electron microscope. Compare to Figure 4.12. 67
4.15 A group of punctured holes in the reflective layer, captured during the playing of the compact disc. 68
4.16 One punctured hole in the reflective layer, captured with maximum resolution during the playback of the compact disc. 69
4.17 Two captures of the same disc. a) cleaned disc b) after touching. The shaded areas mark lower reflection caused by grease and dust. 70
4.18 a) Moiré-pattern in the substrate detected on a badly manufactured disc b) a clean, good disc without moiré-pattern 71
4.19 Demonstrative example of an injection artefact in badly manufactured compact discs (Basler AG, Germany). 71
4.20 A captured overview picture of a badly manufactured disc with lots of defects seen as small dots. 73
4.21 A zoomed-in version of 4.20, showing distinctively the defects, moiré-pattern and a part of the boundary. 74
4.22 A photograph of the surface taken by a conventional optical microscope. 75
4.23 Image capture of the disc with a circular boundary drawn by a ball-pen around reflective layer defects. 76

5.1 Diagram showing interpolation and symbol error rate (crosses) as obtained by the author's software. The straight line represents the interpolation rate, published in [1] as a result of statistical calculations. . . . 84
5.2 Decoder I and II compared in terms of P_{xx} values on a non-bursty channel. Decoder II shows better performance. 85

5.3 Decoder I and II compared in terms of interpolation, click and symbol error rate on a non-bursty channel. Decoder II shows better performance. 86

5.4 Gilbert model 87

5.5 Different probability density functions used for simulations of bursts (area under slope standardised to one). a) bursts caused by a memoryless, non-bursty channel, b) bursts emulating measured distribution in [2] on a clean disc, c) bursts with a ten times higher probability assuming a scratched disc. Curve b and c is reproduced using a combination of random number deviates. 89

5.6 Relation between stretch factor of gap lengths n to obtained error rate r. The error distribution is constant while scaling the gap distribution with factor n to achieve different input symbol error rates. 91

5.7 Decoder I and II compared in terms of symbol error rate and interpolation rate on a bursty channel. Decoder II shows the better correction performance. 92

5.8 Decoder II applied to a non-bursty channel model and one with the distribution presented in paragraph 5.4.2. Towards lower input error rates the difference is evident and can be extrapolated to the deviation of several magnitudes. 94

5.9 Correctibility of equidistant error symbols (EQU) together with background noise. The relative error frequency f is based on the correction rates for background noise only. 100

5.10 Correctibility of a sequence of errors with varying length at the beginning of each frame (ESE) plus background noise based on error rates of background noise only. 102

5.11 Schematic illustration of a shifted sequenced error pattern (ESS) (refer to Figure 5.12). 102

5.12 Correctibility of a pattern with shifted sequences of erroneous symbols each frame (ESS) plus background noise. The relative error frequency is $f > 10$ pointing to a lower correctibility than without shifting (compare to Figure 5.10). 103

5.13 Schematic view of error groups distributed over a frame (EGR).
a) 2 symbols per frame. b) 5 symbols per frame. c) 12 symbols per frame. 104
5.14 Correctibility of groups of error symbols (EGR) plus background noise. For low error rates $r < 0.07$ the pattern is identical to the one used in Figure 5.10. 105
5.15 Correctibility of a pattern with alternating error-free and erroneous frames (EQF1) plus background noise. 106
5.16 Equidistant erroneous frames, the intermediate gap length is variable.
a) one frame distance between one erroneous frame. b) two frames distance between one erroneous frame. 107
5.17 Correctibility of a pattern of alternating error-free and erroneous frames plus background noise. The erroneous frames are filled with alternating error and non-error symbols (EQF2). 107
5.18 Equidistant erroneous frames, the intermediate gap length is variable. The erroneous frames are filled with alternating error and non-error symbols.
a) one error-free frame distance between one erroneous frame. b) two frames distance between one erroneous frame. 108
5.19 Correctibility of a pattern of alternating error-free and erroneous frames plus background noise. The erroneous frames are filled alternatingly with one error and two non-error symbols (EQF3). 108
5.20 Equidistant erroneous frames, the intermediate gap length is variable. The erroneous frames are filled with alternating one error and two non-error symbols.
a) one error-free frame distance between one erroneous frame. b) two frames distance between one erroneous frame. 109
5.21 Correctibility of randomly distributed one-symbol errors (RES) approximating the lower limit of $f = 1$ very slowly. 111

5.22 Summarised overview of four error patterns. The notation "PDF" means bursty background noise is added. The other notations refer to the type of pattern used. Three different groups are recognisable with different types of approximation. The next four patterns are shown in the next diagram, Figure 5.23. 112

5.23 Set of the next four error patterns (continued from Figure 5.22). 113

5.24 The relative error frequency f is standardised to the introduced symbol error rate r. The minimum signifies a good ratio of correctibility to information content. Four error patterns are shown here, the next four in the next diagram, Figure 5.25. 115

5.25 Set of the next four patterns, relative error frequency f standardised to the introduced error rate r (continued from Figure 5.24). 116

5.26 Output error rates of background noise with varying input error rates r if a) a regular watermark pattern is present and b) only background noise is to correct. 118

B.1 Flowchart of the channel encoder implementation for CD audio. 150

B.2 Flowchart of the channel decoder implementation for CD audio. 151

List of Tables

1.1 Standards defining "Compact Discs". 9

2.1 Algorithmic view of decoder strategy I. f denotes the number of erased symbols in a code word. C1 is the inner Reed-Solomon decoder, C2 the outer decoder. 21

2.2 Algorithmic view of decoder strategy II. 21

3.1 A choice of possible error conditions during reading of a CD-ROM/CD-DA (taken from the SCSI-2 specification). 35

5.1 Lowest error probability to detect when decoding $2.5 \cdot 10^7$ frames with 1,10 or 40 symbol errors occurring. One additional symbol error leads to the resulting standard error. 82

5.2 Output symbol error rates $P_{sym,random/bursts}$ of non-bursty, random background errors and bursty background errors, combined with systematically induced symbol errors of two different error rates r_{intro}. The ratio shows that random and bursty noise have different correction rates P_{sym}, as expected, but also that the ratio of correctibility is depending on the rate of induced errors. 98

5.3 After decoding 250000 frames the number of symbol errors n varies depending on the form of noise present. The induced error rate is $r_{intro} = 0.0625$, which is a two-symbol error per frame. 99

5.4 Considering the values derived from Table 5.3 it becomes evident that introducing a two-symbol error per frame does not have the same effects on the output correction rate on bursty and non-bursty background noise. 99

5.5 Fast random number generator for distributing erroneous symbols. . . . 110

Acknowledgement

I would like to express my gratitude to all the people who supported me during my time in Plymouth.

Thanks go to my supervisors Dr. T. Donnelly and Dr. P. J. Davey for their informal and friendly support and the University of Plymouth for giving me this opportunity.

Last, not least of all, I would like to thank all those friends in Plymouth who befriended and supported me during my brief stay in Plymouth and influenced me in their own way.

Author's declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award.

This study was financed by a University of Plymouth studentship.

During the research programme I undertook a course of advanced studies. These include the extensive reading of literature relevant to the research project, and attendance of seminars on signal processing, coding and computing. The work has been regularly presented at research seminars.

Signed .....

Date 28.5.2002

Chapter 1

Introduction

1.1 The Compact Disc

The compact disc (CD) is nowadays a very important medium for permanently storing information. It was introduced in 1980 by Philips and Sony as the next generation successor of the vinyl record. The convenience of use and the immense data capacity predestined it for further development and exploitation.

Soon after its introduction it was employed for other uses such as storing raw digital data. The basic technology behind it proved to be reliable, easy-to-use, inexpensive and extendable. Today CDs and their writable relatives CD-RW, CD-R are everywhere; they are used for distributing and conserving music, for multimedia applications, as a storage medium for computers and as backup medium. Simply speaking, the user is free to duplicate digital data and store it without quality loss on this inexpensive medium.

The successor of the CD, the Digital Versatile Disc (DVD), based on a similar technology with an increased storage capacity, is now gaining commercial momentum

and shows an even greater commercial potential than its predecessor.

There are still certain important issues to be addressed though. The uncontrollable, lossless duplication of digital data, which is in most cases protected by copyright laws, is a worry to the music industry and many software manufacturers.

Protecting digital data from being copied is a controversial topic. Even Philips, the inventor of the compact disc, states in a recent interview [3] that today's copy protection measures for audio compact discs are a nuisance for the consumers and technically not feasible.

The traditional argument for copy-protecting solutions goes as follows. The digital data representing music, program code or other forms of data belongs to the rightful owner who has the copyright of these data. By buying a CD or software (this does not apply to so-called Freeware), the consumer acquires the right to use the data. The use of the data by the consumer is though bound to the license restrictions that apply and does not include free copying.

Being technically in the situation to have the data at his free disposal, the user is able to make copies and redistribute these without technical restrictions. This is certainly not the intent of the copyright holders, thus technical solutions hand in hand with legal regulations try to circumvent this.

The technical side of circumvention of unauthorised copying of audio compact discs consists mainly of simple ideas that violate the compact disc standard. None of the measurements against software piracy are successful either. The latest advances in copy protecting audio compact discs are based on differences of the capabilities of CD rewriter devices to normal audio CD devices. CD rewriters are confused by non-standard manufactured discs. Sooner or later the manufacturers of CD rewriters will

allow for copying of these specially protected, wrong discs as a sales argument.

The broader and more promising picture shows so-called Digital Rights Management (DRM) systems coming up [4]. They are focussed towards locking the digital content away by encrypting it and limiting therefore its distribution to only those who pay. The architecture is still being developed and the further development extends its use to description, identification, trading, protection, monitoring and tracking of all forms of rights usages over both tangible and intangible assets.

Only by taking this global approach, can control over digital contents and rights be ensured. A crucial part is to be played by so-called watermarking and labelling techniques.

Because of their high abstraction level, digital data themselves are perfectly copyable in an infinitive number without the possibility to track them. Distribution of digital data defies control therefore. Obviously it is not possible to imprint additional data, like ownership certificates, instruction for distribution or tracking information on them.

The technique of watermarking gives a means to do exactly this. In general extra information is added in such a way that it does not degrade the quality of the data nor extends the volume of the data unreasonably. This information is hidden to the user and it must be ensured that removing it is a highly difficult procedure. It encompasses, for example, copyright messages, serial numbers or instructions for the use of the data. Copyright notes can help to prosecute the copyright violator, whereas serial numbers serve to identify them.

Adding hidden watermarking data can be targeted at several goals. One also must consider the specific storage medium the data resides on. Watermarking the hardware

medium has similar applications like watermarking the pure data stream. An approach which combines both, hardware and abstract data seems to be most promising.

In this work new ways of watermarking techniques are presented and investigated. The medium is, as mentioned before, the compact disc digital audio, which is the basic format for all other compact disc deviates. The DVD is covered as well, since it employs the same, but a more optimised technology. This work investigates the use of the CD device as a low-cost image scanning (capturing) system, which has been devised and implemented by the author and which primarily makes new watermarking techniques specific to this device possible.

Adding extra data is especially considered with regard to the error correction ability of these devices. By using error correction it is feasible to superimpose data in the form of errors without loss of the user-data. Different methods, ranging from mechanically puncturing compact discs to directly altering the digital data, are investigated.

In this regard the employment of the CD image scanning device as a low cost scanning system is demonstrated. The author compares the image scanning system with electron scanning microscopes, which are much more expensive. The high resolution gained with this cost-effective system opens a variety of new applications, only one of them being secure watermarking.

1.2 The Compact Disc's Vulnerability to Errors

1.2.1 Coping with Errors on Compact Discs

Being manufactured in mass production and having a bit density in the sub-micron range, imperfections, inherent and introduced, of the compact disc make it almost

impossible to store and read information without errors. The advantage of using digital encoding is that error-correction algorithms can be applied to cope with almost any deterioration of the signal. Besides imperfections caused by non-optimum compact disc players, causing mis-tracking or mis-focusing, the need for an error correction system derives mainly from non-optimal production methods and mis-handling of compact discs.

The Compact Discs Standard, as proposed by Philips [5] and specified amongst others by the British Standard (and European Standard) BS EN 60908 [6], takes this into account by specifying the limits under which a compact disc is expected to operate. Due to the need for accepting a non-optimal environment, a sophisticated coding scheme for storing digital data has been conceived which accepts the presence of errors in certain limits as laid down in this standard.

Two types of errors occur: random errors and burst errors. Random symbol errors have no correlation with other errors; when errors occur in a group of symbols they are called burst errors.

1.2.2 Manufacturing Errors on Compact Discs

Optical media manufacture consists of various production steps such as injection moulding, sputtering, protective coating or bonding. These process steps imply possible defect sources which compromise the playability of the media. Most common defects in the production process are:

bubbles: Air-filled holes in the substrate.

black spots: Dirt in the substrate.

bump: Distortion of the reflective layer.

pinholes: Holes in reflective layer.

oil stains: Oily remains on the reflective layer.

lacquer splashes: Remains of lacquer on the surface.

metal-layer scratches: Scratch on the reflective layer.

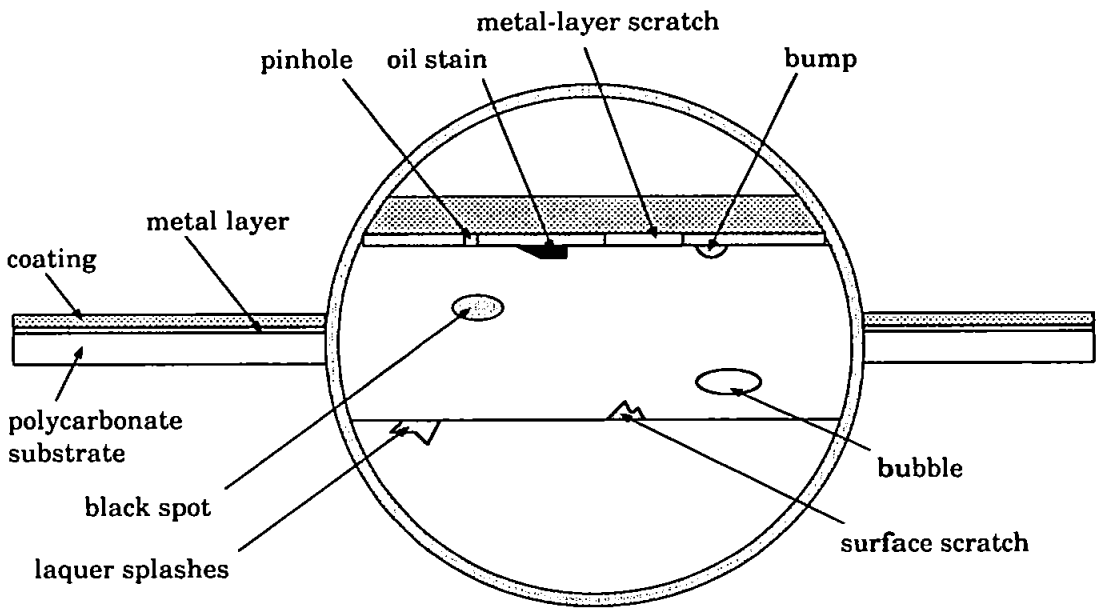


Figure 1.1: Schematic view of defect classification in a Compact Disc.

The various types of defects occurring on a compact disc are shown in Figure 1.1.

The maximum dimensions for bubbles and black spots are $100\mu m$ and $300\mu m$ respectively according to the standard [6]. The minimum distance, measured between adjacent defects (of maximum diameter) along the track, must be at least $20mm$. Conforming with the standard burst errors due to local defects on the manufactured disc shall not induce audible effects for any error decoding strategy.

1.2.3 Blemishes resulting from User-handling

Blemishes which are introduced during use, are attributed to the following:

scratches on the protective layer side: these are permanent damage to the surface. They result usually from careless user handling, but can be cured with commercial CD repair kits. Usually they appear as very thin lines and a number of them grouped together pose problems to the error correction system.

dust, dirt: these are temporary stainings on the surface that can be wiped away quite easily with soap or white spirit.

fingerprints: grease from fingers sticking on the surface of a CD.

1.2.4 Definitions

For a clear understanding of the following chapters, a definition of commonly used terms is given here. Other publications might use slightly different definitions.

sample: The digital signal recorded on a compact disc is sampled in two channels (for stereo reproduction) at a frequency of 44100 Hz with 16bit quantisation. Each 32bit sample value (*audio-sample*) consists of two *mono-audio-samples* each of them 16bit long.

symbol: Each mono-audio-sample is made up of two 8bit *symbols*. A symbol is the smallest addressable unit in the en/de-coding process.

frame: A *frame* is the smallest logical unit in the en/de-coding process. One frame at a time serves as input for the error correction system. It contains 6 audio-samples; in the encoded case 8 error correction symbols are added, which amounts to 588 bits including synchronisation bits, EC bits, merging bits (Figure 3.5).

error: The term *error* denotes a not-intended, wrong value stored in a symbol. It can refer to samples or bigger units as well if explicitly mentioned.

erased, flagged symbol: The Reed-Solomon error correction system receives at its input *erasure flags* (appendix A), one per symbol. They notify the decoder of a possible error on this symbol. Knowing the position improves the error correction ability.

left, right channel: Recordings are done independently on two channels to give the listener a spatial (stereo) impression. Both left and right channels are recorded on the compact disc.

input symbol error rate r: Wrong symbols per second before the error correction process took place.

output symbol error rate P_{xx}: Wrong symbols per second after error correction process took place.

concealment: Detected wrong samples are not played back, but concealed or muted, depending on the method used by the CD-player. Concealment only applies to compact disc audio players.

click: An audible noise in the loudspeaker as a result of a miscorrection or an undetected, not concealed error.

1.3 Compact Disc Standards

Nowadays the term compact disc denotes not only one kind of optical storage medium, but a variety of different standards and specifications. Starting from the standard developed by Philips and Sony [7], described in the *Red Book*, the original specification was extended in many ways to satisfy increasing customer needs. To each of the new

standards a new "coloured book" was dedicated, describing the logical and physical structure of the data and the medium.

Red Book	physical format for audio CDs (a/k/a CD-DA)
Yellow Book	physical format for data CDs (a/k/a CD-ROM)
Orange Book	physical format for recordable CDs
Green Book	physical format for CD-i
Part I	CD-MO (Magneto-Optical)
Part II	CD-WO (Write-Once; includes "hybrid" spec for PhotoCD)
Part III	CD-RW (ReWritable)
Blue Book	CD Extra (occasionally used to refer to LaserDisc format)
White Book	format for VideoCD
CD-ROM/XA	eXtended Architecture, a bridge between Yellow Book and CD-i
CD Extra	a two-session CD, 1st is CD-DA, 2nd is data (a/k/a CD Plus)
FORM-1	2048 bytes of data, with error correction, for data
FORM-2	2324 bytes of data, no ecc, for audio/video
MODE-1	standard Yellow Book sectors
MODE-2	may be of form-1 or form-2
Rock Ridge	extensions allowing long filenames and UNIX-style symlinks
ISO-9660	file layout standard (evolved from High Sierra format)
CD-UDF	industry-standard incremental packet-writing file system
CD-RFS	Sony's incremental packet-writing file system
CD-Text	Philips' standard for encoding disc and track data on audio CDs

Table 1.1: Standards defining "Compact Discs".

Table 1.1 gives a short overview of these standards. Indeed all these formats can be called compact discs, since the design of the physical medium is the same. The specifications laid out in the Red Book standard (CD-DA, compact disc digital audio) form the basis for all other compact disc formats. It describes the physical properties of the medium and goes on to specify the coding of the user data, the basic error correction scheme, EFM-modulation, frame format and sector layout. These are common to all other standards, differences evolve at upper layers, i.e. further error correction is added, certain sector contents have different use and the file system structure is changed or newly designed [8, 9].

Without restricting generality this work only deals with the least common denom-

inator, the CD digital audio. The format layout explained in the following chapters refers to CD-DA and is common to other CD formats as well.

1.4 Contribution to Knowledge

This work presents a new technique based on current compact disc technology to image the transparent surface of a compact disc or additionally the reflective information layer. This technique (image capture technique) has been tested and successfully applied to the detection of compact disc watermarks and biometrical information.

The capture system comprises specialised hard- and software utilising the compact disc technology. Images of human fingerprints on a disc's surface have been captured in high quality.

The author's work also introduces a new idea of watermarking the compact disc medium as part of a future DRM system. It has been shown by the author that watermarked discs can be identified on their digital fingerprints using the novel author's capture system, whilst playing the compact disc normally. Two different ways of implementing mechanical watermarks have been investigated and successfully demonstrated.

Watermark sequences have been tested against their influence on the compact disc's error correction system. Quantitative results regarding the effect of certain watermark patterns are presented. Optimal sequences have been found.

Special error distributions, composed of random and burst errors, as present in user-handled discs were investigated on their correctibility by two error strategies through computer simulation. Conformance with published results in the case of a memoryless channel were found, the work extends the results on error correction to special error distribution functions. These results were used as a basis for predicting effects of

proposed watermarking schemes on the compact disc system.

1.5 Organisation of the Thesis

Chapter 2 gives an overview of the issues, dealt with in this thesis. It presents the necessary background to put it into an historical, scientific context. Work done by other authors tangent to the various fields presented in this work is critically discussed with regard to the author's own work.

In Chapter 3 the experimental apparatus and the respective control software along with the author's own simulating system to backup experimental results is described. The experimental system served as a working demonstration of the capabilities.

The basics to compact disc technology are briefly treated, followed by a discussion of both the experimental and the theoretical system. The software implementation is presented with all its components.

Chapter 4 presents experimental results achieved with the apparatus. Discussions of said results are included in their respective context. Comparison with other common image capture systems are shown, the relevance of these results to the initial issue watermarking and further possible exploration is also demonstrated.

Chapter 5 finally deals with simulation of the experimental technique in order to obtain new insights to the error-correction process and the relevance to new techniques of watermarking.

The thesis is concluded with Chapter 6, reviewing the author's work and discussing it in detail. A paragraph is devoted to future areas of interest.

Chapter 2

Background to the Investigation

2.1 The Compact Disc System - Benefits and Problems

This work incorporates research in more than one area. It is positioned between copy protection research, copyright management, implementing these issues into current hardware of compact disc players and discussing the capability of the new technique.

It has long been an issue for software makers and record labels to restrict the capabilities of the compact discs system to a more controllable medium. Engineering the compact disc has been done without paying much attention to the fact that the data stored on it is perfectly duplicable. But it must be considered that none counted on the wide availability of CD recorders for home consumer use nor has anyone taken into account the falling prices for storage media like hard magnetic discs or the availability of compression formats like MPEG Audio Layer-3 (mp3).

Even the DVD has, according to the opinion of some experts, only rudimentary copy protection standards built in [10, 11].

It is therefore of interest to propose and investigate new approaches for the compact disc system. Due to the number of technologies involved, the system possesses a high extensibility.

As the compact disc system employs a variety of technologies, the author's work also covers many areas, including error correction, watermarking, scanning microscopy, error distributions on compact disc, error correction algorithms and computer simulations.

2.2 Watermarking and Steganography on Compact Discs

As the discussion about copyrights on music, software etc. goes on, ways are needed to enforce related legal issues. Simple copy protection is not always suitable, because copy protection schemes are likely to be broken by hackers. The more widespread and desirable the information content is, the quicker ways are found to circumvent anti-piracy measures [12]. Furthermore the consumer is annoyed by being deprived of his right to make legal back-up copies.

Watermarking and steganography both aim at hiding information [13]. Classical steganography deals with embedding a secret message in a cover message. There are no requirements about the resilience against attempts for removal [14]. Usually, once the covert channel is uncovered, the means for exchanging information through this channel is void. This is opposed to watermarking techniques, where secret information is also hidden, but a possible attack to make the watermark undetectable or even to remove it completely is hindered by certain methods. The existence of a watermark can even

be known to the attacker, but the importance lies in their robustness against attempts to remove it. This is why watermarking techniques are valuable means for hiding information, whenever the cover-data is available to third-parties and the existence of a watermark is known. The information hidden by a watermark is always associated to the digital object to be protected, whereas steganographic methods just hide any information.

If we cannot protect binary data against illegal copying we introduce watermarks to prove the ownership of the data. These copyright messages can serve to prove the ownership of the data before a court, if legal infringements have happened. Data content protected this way are therefore not anonymous anymore, they have a copyright owner and possibly an authorised owner, who, for example, paid for them.

Apart from protecting the ownership by hiding a copyright notice in the data, watermarking can serve two other functions.

The second function is called fingerprinting and is used to track illegal distribution of data. For this purpose different watermarks specific to the recipient of the data are generated and embedded in the data content. The recipient distributing the data illegally can be found by looking up the watermark listed for this recipient. Then it is known that it was him, who gave the copyrighted data away. Here the watermark has to be robust against attacks like in the previous case.

The third function implements a sender identification and authentification and content verification [15, 16]. Two powerful aspects of information can be included in watermarks. The first is error control code (ECC) information, which enables the receiver to make use of error detection and correction systems on the watermarked object. The second combines the possibilities of public key cryptography [17] with watermarking.

In fact sender authentification methods with watermarking can easily be conceived using public key systems.

J. Lee and C.S. Won [18] propose in their work an image authentication and correction system. Briefly speaking watermarks are added to an image in place of the LSB (least significant bit). These watermarks contain the ECC information, which makes it possible to correct the image after alteration. The watermarks though can only be retrieved if the right key is known to the recipient (and thus correction of the watermarked image). The possession of the right key is ensured by the sender encoding the watermarking key with his private key and then by the recipient decoding the watermarking key with the public key of the sender [19].

There are other useful areas where watermarking techniques can be employed. Copy protection systems can be implemented as well. Therefore the copy device must check for the presence of watermarks to decide whether it is allowed to make a copy (or how many) and not.

How are steganographic techniques related to compact disc technology? Since these techniques provide a powerful means for many copyright issues, it is desirable to include them in actual CD technology.

Redundancy of data, which is a prerequisite of hidden data channels, can be found in many places in compact discs. The compact disc system is engineered with a redundancy of stored data of up to 30%. This serves mainly for the detection and correction of symbol errors, for which a highly-optimised storage medium, like the compact disc, is particular vulnerable. Part of this redundancy however can be used for other purposes.

To the author's knowledge there are no publications of specific watermarking techniques for compact disc systems. Since the compact disc system has its own way of

encoding and decoding along with special physical properties and limits, it is therefore of general interest to analyse the compact disc system with regard to this technology.

This work tackles the problem from two different points of view. First symbols of the data stream can be altered to make space for watermarking information. This is done without changing the error correction information accordingly. Thus watermarks are present on the compact disc but do not degrade the real information content due to the error correction ability of the CD device. In this case care must be taken not to exceed the maximum error correction capability. Also locations must be determined where it is the least severe to the error correction to introduce these watermarks. As it is commonly known, bursts (long sequences) of symbol errors can weaken the error correction system much more than randomly distributed symbols. The question therefore is, how a systematic allocation of symbols performs in terms of error correction capacity. This work will suggest some symbol error allocation algorithms which show a good performance. The tool to accomplish this task is a simulation of the channel encoding and decoding process as it is done in real compact disc players. Information hiding is thus simulated while considering a real-world scenario, where the error correction information is partly used up for correcting natural blemishes on the compact disc, as caused by user-handling or failures in the manufacturing process.

Secondly an apparatus for detecting physical watermarks will be proposed in this work.

For generating digital watermarks one does not necessarily rely on modifying the pure data content alone. Other introduction processes are conceivable. Considering the structure of a compact disc, there are many ways which allow the watermarking of the actual physical medium. The benefits can be comparable to watermarking digital data,

as discussed earlier, though with other characteristics. The real advantage evolves when both, watermarks on the digital data and watermarks on the actual physical medium, are combined. As it will be shown watermarking the physical medium is not without effect on the digital data. Both go hand in hand and the error correction process further improves the number of possible applications.

In this sense introducing watermarks means damaging or puncturing the physical medium. If it is done in-between the limits of the error correction capability, it is without effect on the restored (audio) signal. In order to detect such watermarks, the compact disc player must be enabled to process these artificial blemishes.

The advantages of this are numerous. Due to the connection between errors in the digital data content and errors in the actual physical medium the two of them are bound together. Thus it is ensured that the digital data belong to its storage medium. This feature can be part of a DRM system, although it must be ensured that compact disc players have this capability built in.

In general the fingerprinting feature permits the marking of compact discs individually and uniquely; applicable solutions depend on the way it is implemented.

Implementing it into actual CD technology is not hard at all. The following chapters will give a thorough investigation and discussion. The technique for detecting additional watermarks on the medium itself relies on the principle of a scanning microscope for which the CD player is used. It will be shown that besides detecting watermarks or other distinctions, the technique is able to scan compact discs with high resolution thus making it possible to increase data capacity by adding graphical information on "top" of the digitally-encoded data.

2.3 Scanning Microscopy

Apparatus for scanning discs are commonly used in the quality-checking process in compact-disc manufacturing. The scanners are part of the production line and provide process information for quality optimisation. Being specialised for CD-ROM, DVD,CD-R, CD/DVD-RW/RAM, they detect all kinds of manufacturing errors, such as defects on surface, reflection layer and substrate, tilt irregularities and resin thickness [20].

Commercially available basic systems cost about £3000 sterling. These systems use customised hardware, different from a common CD-device. The CD is spun up to 1000rpm, line scanning it with a matrix camera, photodiode or interferometer. Measurement time extends to 10s maximum. A maximum spatial resolution for defects of $11\mu m$ is achieved (specifications according to [20]). Additionally, bundled software characterises the type of defects and gives information in order to optimise the production process. Various other commercial systems are also available [21].

The author's work however employs an unmodified readout head, reducing the costs substantially. The drawback is a slower scanning time, which depends on the reading speed of the CD device. Using modern CD-devices the scanning process can be speeded up by 52 times compared to early single speed (data transfer rate 150 kB/s) devices, i.e a 65min CD is scanned in 1.4 minutes. The resolution across tracks though is better ($1.7\mu m$), since each adjacent track is scanned individually. The resolution along tracks is dependent on the hardware used, in particular on the analog-digital converter and on the speed of the CD device, which in this case limits the resolution to about $15\mu m$.

In [22] a scanning optical microscope is presented. The setup uses a conventional CD reading head, where the laser is replaced by the end face of a single-mode fiber. Employing automatic focus control the microscope is able to scan over curved objects

having a spatial cut off frequency of 1.1 line pairs/μm. This system uses self-built scanning mechanics which enables it to keep track even without the existence of pits on a CD.

For the author's application using the built-in tracking control into the CD device was more appropriate, since the interest lies specifically in the readout-signal, which the CD-player processes. The low-cost factor and the fact that very few modifications are necessary in order to add new functionality to CD devices, makes this technique attractive for a future element in copy protection and copyright protection systems.

The scanning system's setup will be presented in Chapter 3. A discussion of the possibilities under special consideration of watermarking purposes (different fingerprinting techniques) and comparison to other image capture systems will be presented in Chapter 4.

2.4 Error Correction Simulations

Format and encoding procedures for compact discs are defined in various standards [6, 23, 24, 25]. Computer simulations can give information about behaviour and limits of the coding process. As mentioned before (1.3), CD-DA and CD-ROM channel coding are different. CD-ROMs employ additional coding in order to meet the more strict requirements for data storage, where interpolation and muting of unrecovered errors from CIRC cannot be tolerated. The additional coding comprises a second layer EC system (CRC) and another step of scrambling for data whitening [26].

To the author's knowledge computer simulations of the compact disc channel have been published in only one other article. In this work J.D. Roberts et al [27] simulate the channel coding of a CD-ROM system. Information about the correctibility of burst

errors according to their position relative to the CD-ROM sector boundaries have been obtained. Their research shows clearly that burst errors (i.e. scratches on the surface) achieve a higher correctibility if they lie near the centre of a CD-ROM sector compared to the edges of a sector.

The author's work extends the work of [27] by applying the simulation to special error patterns in conjunction with watermarking purposes. It investigates not only one symbol error burst but concentrates on predicting the effects of complex error structures. Errors, like defects in the substrate or dirt on the surface, are handled by considering certain error distribution already present. These error distributions have been measured in publications of other authors and are used in this work. Additionally different error decoder strategies have been considered as there are different algorithms used in compact disc players nowadays.

The encoding procedure complies in the main parts with [6] (For differences see Section 3.3). Decoding is not a subject of standardisation. Published algorithms for error decoding strategies [1, 28, 29] have been used along with developed algorithms for interleaving and eight-to-fourteen (EFM) decoding.

A number of different decoding strategies are presented in [29, 28]. The author's software implements four of them. One of these four has been chosen to be employed in any simulation run. It was first introduced in 1995 in [30], thus being a relatively modern algorithm. The schema is outlined in Table 2.2 and in the following named "strategy II". A slightly more simple one, first published in 1982 in [1], has been tested as well, mainly to make a comparison between both of them. This algorithm, referred in the following by "strategy I", is outlined in Table 2.1.

C_1 Decoder

```
IF single-error or zero-error syndrome
THEN modify at most one symbol accordingly
ELSE assign erasure flags to all symbols of the received word.
```

C_2 Decoder

```
IF single-error or zero-error syndrome
THEN modify at most one symbol accordingly
ELSE IF f>2
     THEN copy C2 erasure flags from C1 erasure flags
     ELSE IF f=2
          THEN try two-erasure decoding
          ELSE IF f<2 or two-erasure decoding fails
               THEN
               assign erasure flags to all symbols of the received word.
```

Table 2.1: Algorithmic view of decoder strategy I. f denotes the number of erased symbols in a code word. C1 is the inner Reed-Solomon decoder, C2 the outer decoder.

C_1 Decoder

```
IF single-error or zero-error syndrome
THEN modify at most one symbol accordingly
ELSE IF f>2
     THEN copy output C1 erasure flags from input C1 erasure flags
     ELSE IF f=2
          THEN try two-erasure decoding
          ELSE IF f<2 or two-erasure decoding fails
               THEN
               assign erasure flags to all symbols of the received word.
```

C_2 Decoder

```
IF single-error or zero-error syndrome
THEN modify at most one symbol accordingly
ELSE IF f>2
     THEN copy C2 erasure flags from C1 erasure flags
     ELSE IF f=2
          THEN try two-erasure decoding
          ELSE IF f<2 or two-erasure decoding fails
               THEN
               assign erasure flags to all symbols of the received word.
```

Table 2.2: Algorithmic view of decoder strategy II.

2.5 Measuring Error Distributions

Compact discs are not free from errors. The characteristics of a compact disc in terms of present impurities and blemishes can be shown by evaluating an error distribution function. The distribution function showing the probability of occurrence of errors with a certain length of symbols is called *error-burst statistic*, the distribution for the intervening gaps between errors is called *good-data gap statistics.*

Introducing additional information means adding an error distribution function on top of the error distribution already present on a disc. Therefore a simulation has to know about the distribution function.

There are several publications dealing with this task. In this work published error distributions for surface defects have been employed. In [31, 32] measurement hardware is used to generate error statistics. The custom-designed, PC-based analysis tool, presented in [32], examines the serial data at the input of a CD player's CIRC block decoder prior to any EC decoding.

2.6 Performance of Compact Disc Player's Decoder Strategies

Part of the author's work deals with performance issues of certain decoder algorithms.

Performance assessments of the decoding process have been published in several articles. Early work [1] assumes a memoryless channel and decoding strategies, which were available at the time. Interpolation and click probabilities $P_{interpolation}$, P_{click} in their functional dependence to the input error rate r were derived using a mathematical probability model of the decoding strategies. The bursty nature of the input is not

considered as it is with the output of the interleaver in-between the two decoder stages.

In order to obtain measured error-burst and good data-gap statistics on compact discs, hardware has been developed in [31] as previously noted. The next step is to evaluate $P(n,r)$, the probability of having r erroneous bytes in a block (i.e. codeword) of length n bytes at the input of the C1 decoder. This was done in [2]. A state model, based on a Markov chain with a finite number of states, was proposed, characterizing the compact disc channel. Calculations based on this model lead to a recursive formula to calculate $P(n,r)$ from state transition probabilities derived from measured burst and gap statistics.

The distribution $P(n,r)$ represents the occurrence of burst errors. Based on this more realistic assumption calculations in [29] have been done to estimate the statistical probabilities that characterise the data symbols at the output of the CIRC decoder. Assessment and comparison of five different decoding strategies have been done in [29] by deriving analytical expressions for the following terms:

P_{00}: The probabilitity of encountering a correct byte with no attached flag (a good byte; not erased).

P_{01}: The probability of encountering a correct byte that is flagged (a good byte; erased).

P_{10}: The probability of encountering an erroneous byte with no flag (an error; not erased).

P_{11}: The probability of encountering an erroneous byte that is flagged (an error; erased).

Evaluating these expressions, interpolation and click probabilities ($P_{interpolation}$,

P_{click}) for a bursty channel are also obtained, leading to a performance comparison of different decoding strategies. This work will continue to use aforementioned expressions for the error probabilities.

Another analysis of error decoder behaviour is done in [33]. Three simple error-correction strategies used by compact disc players have been compared with respect to random burst errors of specific length. For simplicity the authors assume that the number of byte errors in the input codeword is always a multiple of l, a positive integer. By varying l from 1 to 32 the random to burst error performance of the various strategies is obtained.

The author's work makes a more realistic assumption on the input errors. Input errors obey a certain probability function, measured and published in previous works. Furthermore all interleaving and scrambling stages during signal decoding are considered, which apart from [27], none of the former published work does. The author's work also distinguishes between erasure interpolation situations in the output and divides the result in certain error conditions, as described earlier in this Chapter.

It should be noted that in the above mentioned publications only probabilities of the occurrence of errors at the input of the C1 decoder are considered. As the author's work deals with certain positions of artificially introduced errors, probability calculations are not sufficient and call for a different approach, which will be presented in this work.

Also the fact that all three stages of interleaving and scrambling in the decoding process are not treated elsewhere, gives rise to the idea of using computer simulations to model all steps involved in decoding the data from compact discs.

An accurate computer simulation including the most detail as possible is crucial for obtaining reasonable results for the analysis of watermarking schemes. Section 3.2

introduces the computer simulation of the channel model and Chapter 5 discusses the findings along with the applicability to watermarks.

Chapter 3

Experimental Apparatus and Techniques

3.1 A CD-based Image Capture System

3.1.1 Introduction

A novel system for image capturing using a common compact disc player has been conceived and built. Special software and a few additional hardware components make it possible to search for any kind of defects on compact discs, to scan the surface of a compact disc and generate two-dimensional images of scratches and possible water-markings. The system is low-cost and gives a resolution comparable to an electron microscope and enables a secure way of copy protecting CDs.

The experimental setup comprises an IBM compatible PC (Pentium II, 350MHz) equipped with an internal data acquisition card (PCX 312) and an external CD-ROM device (Toshiba XM3301, XM4101), connected via SCSI interface to the PC. Control and evaluation software written in "C" was used to drive the acquisition card as well as

the CD-ROM device. This software writes the recorded data to files on the computer's hard drive. These are processed by a second set of software programs. The computer runs the Linux [34] operating system, which simplifies programming and real-time data acquisition.

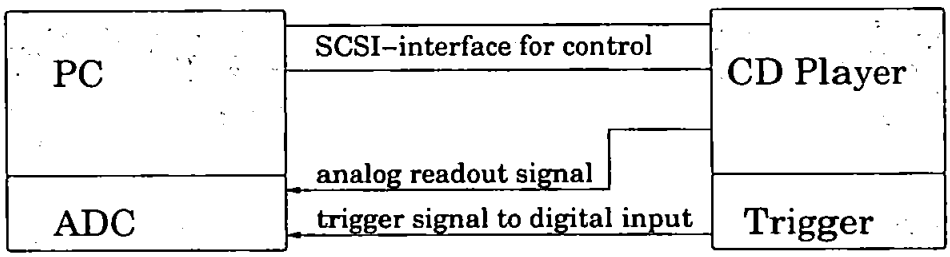


Figure 3.1: Structure of the experimental setup

3.1.2 Basic Construction of a Compact Disc Player

For a clear understanding of the new technique the basic construction of a CD player will be explained in the following.

The read-out process adopted in optical disc systems is that of a scanning microscope. Under the readout head the compact disc spins with constant linear velocity (CLV). The speed varies from $1.2\frac{m}{s} - 1.4\frac{m}{s}$, depending on the compact disc.

There are several types of readout heads, generally they host the laser-diode, photo detector, the optical system, consisting of objective lens, beam splitter, diffraction grating and a magnet with a two-axes device for focus control.

The compact disc is scanned along the tracks by a focused semiconductor laser beam. The focal point lies on the reflective layer of the CD and is about $1.7\mu m$ in diameter. The surface spot is $0.8mm$ in diameter.

The laser beam is split up by a diffraction grating, generating two side beams, one a bit further ahead on the track, the other one following.

Six photodiodes detect the reflected laser beam. Two of them detect the side beams and the remaining four detect each a quarter of the main spot. This construction is utilised by servo circuits, which are responsible for keeping the readout head on focus and on track. Four servo circuits are used (Figure 3.2).

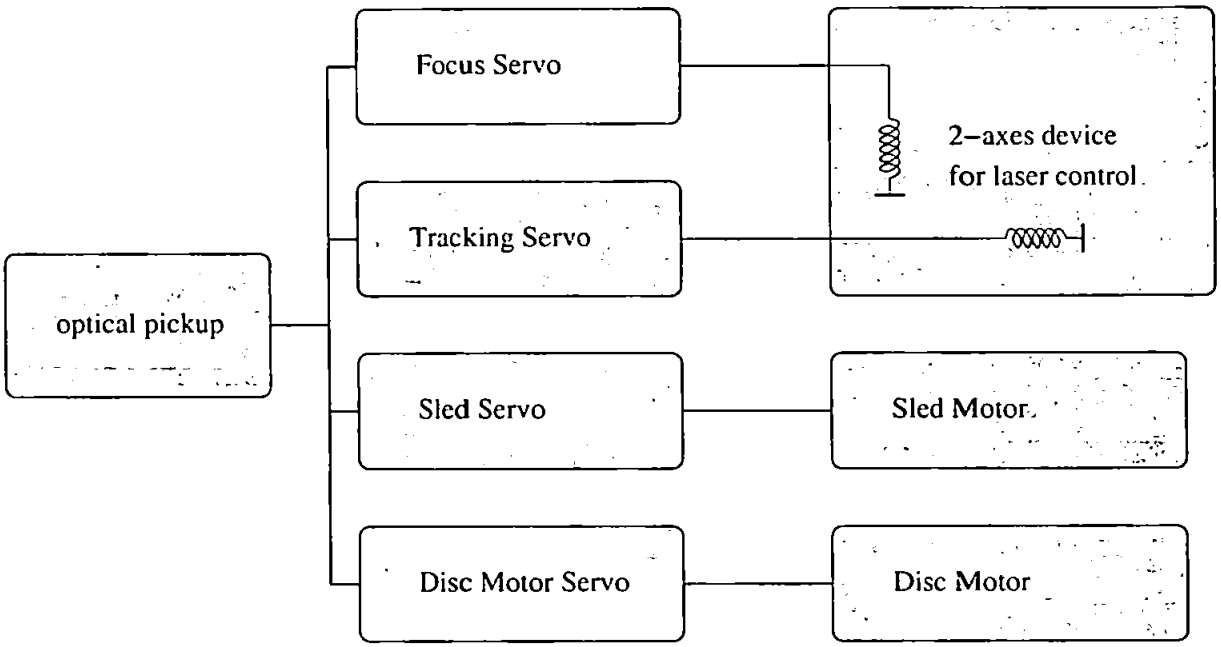


Figure 3.2: Servo circuits in a compact disc player

The tracking servo and the sled servo are controlled by the side beam photo detectors, the focus servo is controlled by a combination of the four main spot detectors.

The servo units are of importance, because they have a limiting character on the scanning process. On some occasions it can happen that the scanning is interrupted because one of the servos loses the track. This happens mainly when the reflection of the laser signal is deteriorated enough to cause the servo electronic to fail.

3.1.3 Presentation of the Novel Capture System

An image capture system has been built and is presented here. Results are presented in the next Chapter. It demonstrates all the functionality needed to reproduce exact

visual images of compact discs and for watermarking applications.

There are two main causes of errors in the read-out process: defects in the reflective layer and disc surface blemishes. The latter comprise surface scratches and/or surface deposits such as dust or inadvertent fingerprints. Defects in the encoded layer are usually caused during the stamping of the disc and are literally holes in the layer. These commonly vary in size from about $10\mu m$ to $50\mu m$ and are often visible to the naked eye by holding the disc in front of a bright light. Each disc's distribution of these defects is unique thus forming a watermark.

Defects result in the loss of encoded bits. Whatever the source of error, provided the latter are not too excessive, a Cross Interleaved Reed-Solomon Error-Correction Code (CIRC) employed in the CD system, corrects them and the "missing" data are recovered [35]. When the laser-beam scans a defect the reflected signal degrades. Additionally, if an error in the data stream is detected, the error-correction procedure is invoked.

Software has been developed which interprets both the degradation of the reflected laser signal and the operation of the error-correction code as a measure of the shape and size of the defect thus permitting this information to be captured. Not only can this technique be used to capture natural defects but also "features" which have been purposely introduced may be captured. These include information introduced on the surface of the disc, such as intentional finger prints and characters plus those introduced when the disc is stamped at manufacture, i.e. coded information in the form of holes placed in the encoded layer at certain points.

The software is supported by a compact disc drive, which has been slightly modified. The additional requirements are minimal: a standard compact disc drive with SCSI interface and an analog-to-digital converter (ADC) card for the PC. The only

customised hardware needed is that to index the PC software to the data on the disc. This comprises a photo-cell device which provides a trigger pulse to the PC once every revolution of the disc (Figure 3.1).

The analog signal of the reflected laser is taken straight from the PCB on the CD device. It carries the amplified output of the player's photo-detector. It is fed into the analog input line of the ADC. The trigger pulse is fed into the digital input line of the ADC. This setup structure is sufficient for the software to successfully sample and evaluate the incoming data.

The developed software package consists of two components: the data-acquisition software and the data and image processing software. The data-acquisition software takes control of the device by sending SCSI commands via the SCSI-interface. Certain music tracks or sectors on the CD can be selected.

Simultaneously, the ADC is polled and the occurrence of a trigger pulse, which determines the beginning and the end of a revolution is checked. The sample values are taken at regular time intervals.

The operating system, a Linux system, is configured to support real-time data acquisition. A minimum selectable time interval of $12\mu s$ is achieved leading to a spatial distance of the sample points of $15\mu m$ along the track. This depends on the computer speed used and the conversion time of the ADC. A faster PC and ADC will give a higher resolution. In order to check if the CD is played correctly, even in the presence of surface defects, status messages reported back from the SCSI device are logged by the software.

After accumulating the raw data specially written image-processing software is applied to generate scalable pseudo-colour images for storing in common picture-file for-

mats.

A photograph of the experimental image capture system is displayed in Figure 3.3.

Figure 3.3: Structure of the image capture system. The photo shows the opened compact disc player and the trigger circuitry.

3.1.4 Necessary Hardware Alterations to the Standard CD-Device

Compact Disc Player Models Employed

Two different CD-ROMs have been used. Both are manufactured by Toshiba, model XM3301 and XM4101. The latter is the successor of the first one. Using two different models was necessary to verify the deviations among different devices. Both of them have got similar circuitry. They differ in the construction of the optical head as well

as in the way they are mounted together. Both of them are external drives and run on single speed with 150 kbyte/s data transfer rate connected via SCSI to the computer.

"Once-around" Trigger

In order to match the digitised data from the compact disc device to an actual location on the disc, a "once-around" trigger has been added. It comprises a LED-detector combination providing a signal pulse each time the compact disc spins around. The structure is fitted on top of the player next to the disc. A paper stripe stuck on the CD and passing through it, generates a trigger pulse, which provides spatial information to the sample data. The paper stripe is about $5mm$ wide, which means a sector area of about 5° is not captured, although experiments show that this can be reduced to $2mm$, with the risk of losing some trigger signals, if necessary.

The LED/photodiode combination is part of circuitry, which hosts an amplifier to interface the output signal of the photodiode to the digital input line of the data acquisition card. The schematics is shown in Figure 3.4.

Data Acquisition Card

The data acquisition card, PCX 312, is a high performance multifunction internal AT-slot card, manufactured by MA Instruments. Its features, relevant for this work, are:

- 12 bit ADC with $3\mu s$ conversion time
- 16 single ended analog inputs
- programmable input gain
- triggering of AD conversion from software and external clock source

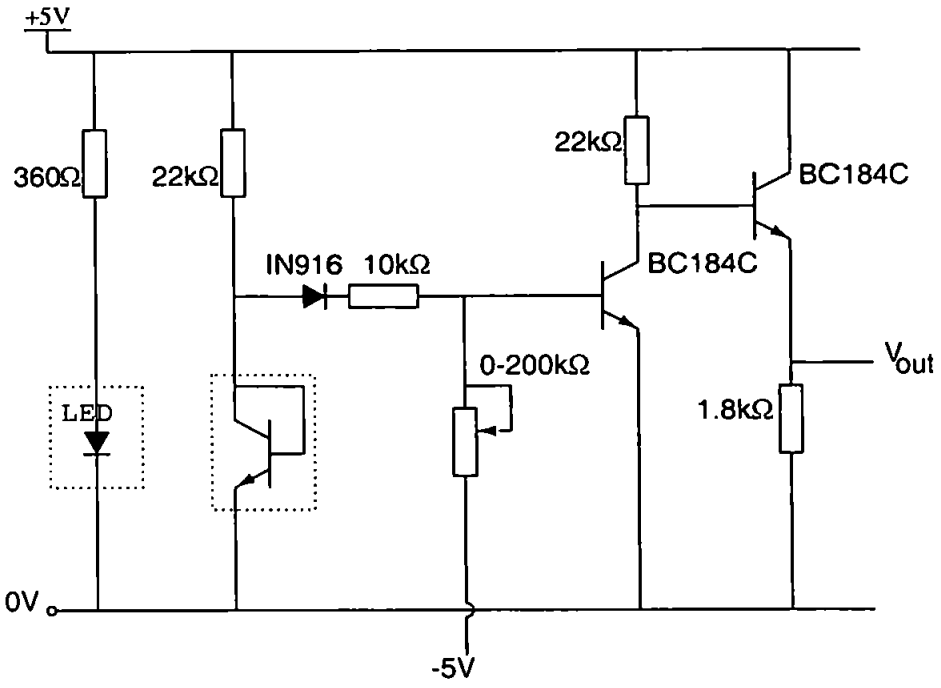


Figure 3.4: Schematic diagram of the interface trigger to ADC.

- 16 bits of digital input provided by HCTMOS device

The digital input line of the ADC is fed with the signal coming from the once-around setup. The analog readout signal coming from the CD device is inputted to the analog input line of the ADC. The PCX 312 does not have an on-board memory, so the computer processor needs to read out the digital value immediately and in equidistant time intervals after conversion.

3.1.5 Control and Data Acquisition Software of the Capture System

The software application for data acquisition and controlling the measurement is written in C and compiled with the GNU C-compiler [36]. The program is command-line driven and performs the following basic tasks:

- driving SCSI and IDE CD-ROM devices
- playing CD-DA and CD-ROM
- controlling the ADC
- SCSI-error logging
- soft real-time data acquisition with timing information

The ADC's control functions and the digitised data are accessed by inline assembly port-commands.

The SCSI-device is controlled over a *generic SCSI-interface*. It provides user-level programs with access to the SCSI-device, thus enabling full control of the devices by sending SCSI-commands. An SCSI generic packet device driver device is accessed by *write()*ing SCSI commands plus any associated outgoing data to it; the resulting status codes and any incoming data are then obtained by a *read()* call.

SCSI-commands with parameters [37] used include:

PLAY AUDIO(12): The PLAY AUDIO(12) command requests the target to begin an audio playback operation. The logical block address parameter specifies the logical block at which the audio playback operation shall begin. The transfer length parameter specifies the number of contiguous logical blocks that shall be played

READ(10): The READ(10) command requests that the target transfer data to the initiator. The most recent data value written in the addressed logical block shall be returned. The logical block address parameter specifies the logical block at which the read operation shall begin.The transfer length parameter specifies the number of contiguous logical blocks of data that shall be transferred.

START STOP UNIT: The START STOP UNIT command requests that the target enable or disable the logical unit for media access operations.

Commands having no output data or commands completing with a CHECK CONDITION status report status information back through the sense-buffer. This buffer provides the user-application with information about the cause of a SCSI-command failure. Particularly useful is the *additional sense code* (ASC) and the *additional sense code qualifier* (ASCQ). These combinations, about 140 altogether, give an exact statement about the internal problems of the device. Some of the more relevant error conditions for this work are listed here.

ASC	ASCQ	description
09	00	TRACK FOLLOWING ERROR
09	01	TRACKING SERVO FAILURE
09	02	FOCUS SERVO FAILURE
09	03	SPINDLE SERVO FAILURE
11	00	UNRECOVERED READ ERROR
11	05	L-EC UNCORRECTABLE ERROR
11	06	CIRC UNRECOVERED ERROR
15	00	RANDOM POSITIONING ERROR

Table 3.1: A choice of possible error conditions during reading of a CD-ROM/CD-DA (taken from the SCSI-2 specification).

The author's software logs time, location and error condition into a separate file in case a CHECK CONDITION has occurred. Since the SCSI-commands address sectors (98 small frames) as the smallest data unit, the error conditions refer to sectors 3.2.2.

By means of this logging facility it is possible to find the locations of occurring errors due to severe defects causing the error correction system to jump in without having to capture all digitised data.

This logging facility is of importance when it comes to determine the error locations for evaluation.

Due to the lack of an on-board memory on the ADC, it is necessary to maintain an accurate timer clock. Polling the ADC is done at equidistant intervals of $12\mu s$.

The ADC returns two 8-bit sample values for each conversion, whereby only the lower 12 bits are relevant, due to the 12 bit conversion in the ADC. The sample data are stored into an array and written after each revolution of the CD to a file on the hard drive.

In order to ensure correct timing, the control software is associated a soft real-time scheduling algorithm of the operating system (Round Robin Scheduling). This means whenever this process becomes runnable, it is granted a pre-emptive right to suspend other processes. For the time the software runs, it is necessary to have a command shell available, which runs at a higher static priority, in order to keep control over the computer. The interval the software is accessing the ADC is controlled by the *nanosleep(2)*-function, which performs busy waits with microsecond precision if the program is scheduled under a real-time policy.

Furthermore the OS runs under single-user mode without network services to ensure that there are as few interruptions to the sampling process as possible.

Simultaneously to playing the CD and receiving the sampled data, the occurrence of the once-around trigger signal is checked. The software checks if one or more trigger signals have been lost. In this case, the sampled data are split up into equal lengths before writing it to the hard drive, simulating the occurrence of the trigger.

The number of the sample values varies from revolution to revolution because of speed variations in the read out loop and variations in the arrival of the trigger signal.

3.1.6 Capabilities of Capture System

A polling time of $T_s = 12.1\mu s$ is achieved. This leads to a sampling frequency of $f_s = 82.6kHz$. The polling time is derived from the measured time for one revolution (excluding paper stripe) divided by the number of counts; each count executes one 16-bit ADC poll. Stopping the time is done by a call to the OS function *gettimeofday(2)*. It queries the internal system time (clock ticks) plus the hardware counter. The hardware counter is driven by a quartz oscillator. The internal system time is set every time the counter reaches the value of a programmable latch. Then an interrupt is generated, the counter is reset and the clock ticks are increased by one.

The duration between consecutive polls is determined by the speed of the computer and the implementation of the polling loop. Since the implementation is done in C, optimisation could be possible if using assembly code. The faster the computer (depending on type of processor and clock rate), the shorter the polling time is.

Improving the sampling period is also possible by using an ADC with on-board memory, the sampling rate would only be depending on the conversion time of the converter. This could improve the results by about ten times.

Every $T_s = 12.1\mu 2$ a sample value is taken. Consecutively the spatial distance on the CD along the track between two samples can be calculated as $L_{polling} = v_{disc} \cdot T_s = 14.5\mu m$.

The whole measuring process is based, and depends on, an accurate timer clock. In order to know the relative timer error, the time needed for playing 30 revolutions of three different tracks of one CD has been timed. The standard deviation amounts in all three cases to $\sigma_T = 1.1 \cdot 10^{-6}s$. With the time stopped being around $T = 0.15s$ the relative time error for one revolution amounts to $\frac{\sigma_T}{T} = 7.3 \cdot 10^{-6}$, which is about the

accuracy of a quartz clock.

3.2 Modelling the Error Recovery Process

3.2.1 Experimental Apparatus versus Simulation Techniques

Not only are natural random defects of importance for this image capture system. But one important application lies in the ability to detect defects introduced on purpose. This for example can serve for watermarking purposes where a second information layer is introduced artificially in certain points or locations.

The effects of this kind of introduction has not previously been investigated. It is not known if and how the error correction system reacts to special code patterns serving as additional watermarks.

In order to be able to predict the reactions of the compact disc player in terms of error correctibility to artificial labeling (watermarking) information, the error introduction process and the compact disc player's channel model have been simulated by the author's software. The software was written with a full simulation of the compact disc's channel coding in mind. This simulation enables a reproducible examination of the experimental technique and reliable results.

The following Section gives an understanding of the principles of the channel coding for compact disc players, which in a later Section is used to model the error recovery process including all necessary steps of the channel model.

3.2.2 Basics about the Channel Coding for Compact Discs

It is obvious that the digitised music signal cannot be stored directly onto the optical medium. Physical and electronic properties of the recording and playback system determine the requirements for the optical channel.

Both stereo channels of the music are sampled simultaneously at 44.1 kHz. The audio samples are linear encoded in a 16-bit 2's complement format. After adding error correction symbols and control words to the data stream, a run-length-limited code, called *eight-to-fourteen modulation (EFM)* is used to meet the requirements of the channel, which are

Clock Content: It must be possible to regenerated the bit clock from the readout signal itself by detecting the pit edges. Therefore the maximum run length must be as small as possible.

Low-Frequency Contents: Low frequency disturbances of the readout signal, caused by dirt and fingerprints, can be filtered out, provided that the signal's low-frequency content is zero. Therefore the longer the minimum runlength is, the better it is.

Error Propagation: Error propagation of the modulation system must match the CIRC EC system in order to be as small as possible.

EFM is a modulation method based on a 8-to-14-bit conversion. With EFM an 8-data bit symbol is mapped onto 14 channel bits. Accordingly the error propagation of EFM is also limited to 8 data bits, thus satisfying the requirement that it should be as small as possible and matching the CIRC EC system. The code is generated in such a way that the minimum distance T_{min} is 3 channel bits and the maximum distance

is 11 channel bits. It therefore produces (2,10) run-length-limited (RLL) channel-bit sequences [38, 39]. The conversion is done by a table lookup.

It is also necessary to add at least 2 channel bits for connecting these patterns without violating the T_{min} constraint. In order to increase flexibility, EFM has 3 selectable channel bits for merging consequent symbols, enabling the suppression of low-frequency content of the frequency spectrum (Figure 3.5).

Finally each positive-going pulse is converted into a single transition, which gives the resulting signal the minimum and maximum length of 3 clock periods ($T_{min} = 3T$) to 11 clock periods ($T_{max} = 11T$) respectively (NRZ to NRZI (non-return to zero inverted) conversion).

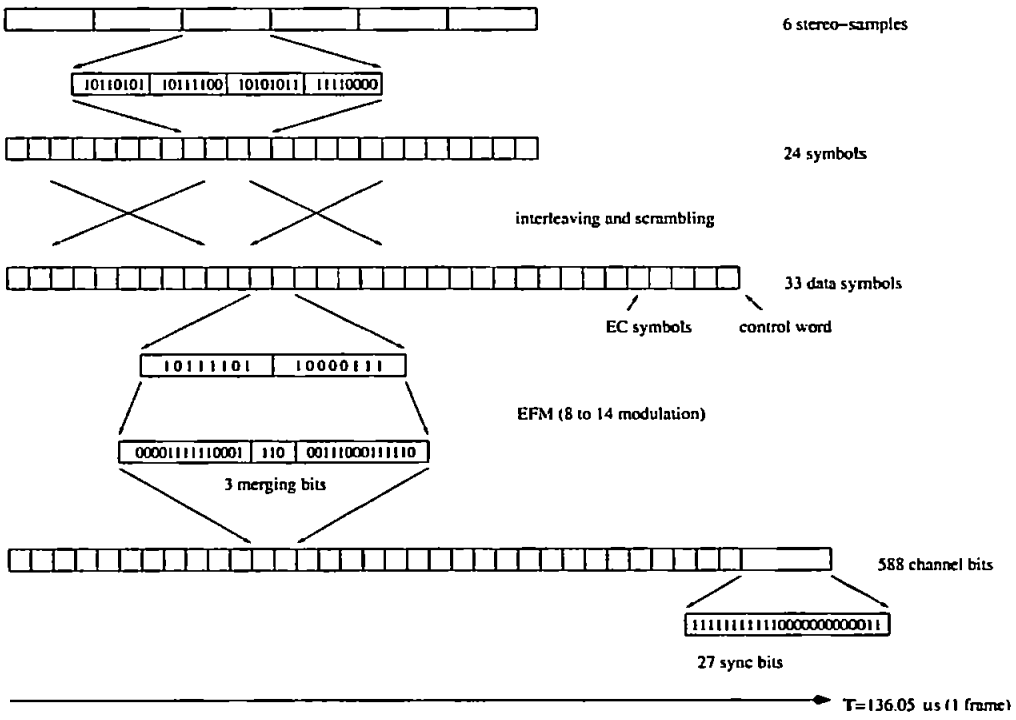


Figure 3.5: Bitstream of the encoding system of the Compact Disc.

According to the standard [6] 24 bytes of music data make up one *small frame* containing 588 channel bits. Each small frame accommodate 4 bytes of EC symbols, 1 byte for additional information, called *control word* and 24 channel bits synchronisation

pattern at the beginning (Fig 3.5). In the following "small frame" is called "frame".

Due to a sampling frequency of $f_s = 44.1\text{kHz}$ with 6 16-bit stereo samples of music per frame, it takes $136.05\mu s$ to play one frame with a final channel bit rate of $4.3218 \cdot 10^6 \frac{bit}{s}$.

98 frames build up one *sector*. The subcode of one sector constitutes a complete unit and supplies amongst others timing information to the CD player. CD-ROMs make use of sectors as a higher-ranking data unit by adding a second layer decoding structure.

The Interleaver

Applying only error correction algorithms to the digital data is not enough. In order to cope with large burst errors caused by macroscopic surface impurities, spreading the original sequence of data during recording over multiple frames into a different sequence improves the error recovery process. This is called interleaving. The reverse process during playback is called deinterleaving. Error correction and interleaving on the CD system are alternatingly applied. This bonding makes it a complicated process with powerful properties.

The core is a (28,112) cross interleaver between the two stages of Reed-Solomon encoders. This cross interleaver falls into the class of product codes but features a variable interleaving delay for each symbol. The resulting convolutional structure usually performs better than a conventional product code. The structure is shown in Figure 3.6.

The interleaving, together with the scrambling process, distributes the data in a defined way. Likewise the burst errors on the surface are broken up into smaller parts,

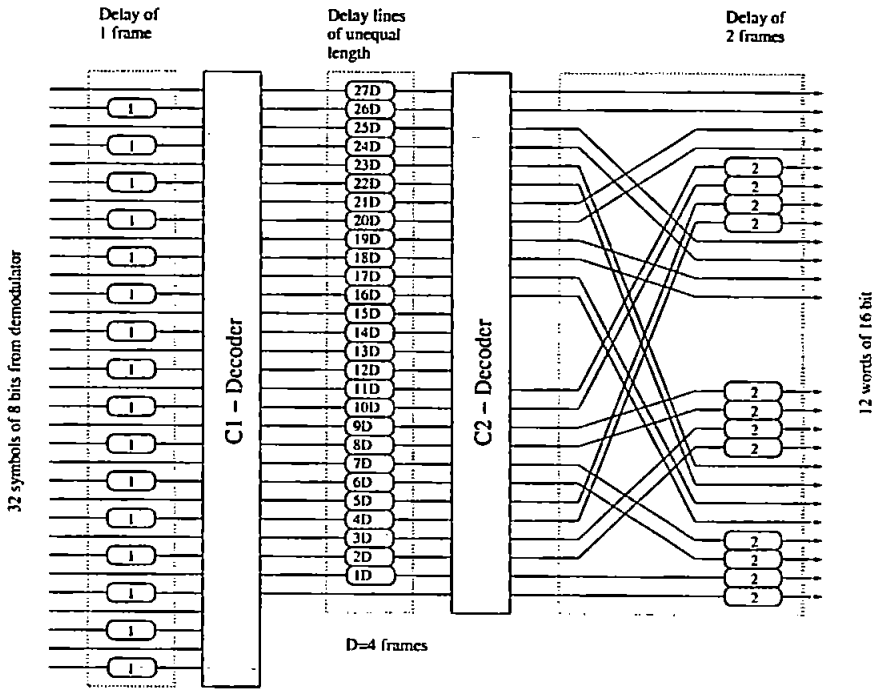


Figure 3.6: CIRC decoder structure

so chances are higher that the number of errors in an error correction block doesn't exceed the limit for correctibility.

3.3 Software Implementation of the Channel Coding and Decoding

3.3.1 Introduction

This Section presents and discusses the counterpart of the experimental apparatus which simulates natural and artificial errors. It implements in software all the steps of the compact disc channel coding system, including Reed-Solomon codec, EFM-modulation, interleaving and scrambling. Additionally functions for control and verification of the software have been added.

The software is completely written in C++ which gives the necessary performance to execute a large number of iterations compared to interpreted languages. The object-oriented approach leads to a more structured source code. For every major en-/decoding task a class was created to provide the necessary functionality and data structure.

Care has been taken to make the source code as portable as possible. The software has been running simultaneously on a HP machine running HP-UX and a Intel machine running Linux.

3.3.2 Implementation of EFM-Modulation

EFM-modulation and demodulation and NRZI conversion is done in a separate class. The translation tables are generated in the class constructor. An algorithm was developed to generate an own table according to the requirements of a run-length-limited code outlined in [38, 40, 41, 42]. Secondly the official table found in [6] can be used and has been employed throughout the work.

Comparing error rates from both tables a difference of 0.2% is revealed in favour of the official one. This difference is not of importance and can be attributed to natural local variations in the random number sequence. The use of the correct EFM table comes into use when dealing with non-random error values.

During encoding merging bits have to be chosen in order to separate the EFM code words and satisfy the code constraints. Three bits allow for a free choice of one bit in most cases, which is used for minimizing the power at low frequencies (DC control). An algorithm based on [38, 42] has been implemented. This algorithm generates a sequence of three merging bits considering the maximum and minimum run-length of

the neighbouring EFM symbols and suppression of DC content. Since the merging bits do not contain any information they are skipped by the software's demodulation stage.

The decoding task encompasses conversion from NRZI to NRZ, skipping merging bits and converting the EFM symbols back to 8bit symbols. If an illegal 14bit EFM symbol is encountered, the function sets appropriate erasure flags to help error decoding.

The encoded frame format consists of 32 symbols. One symbol comprises merging bits and the EFM-code symbol. The 24 synchronisation bits at the beginning of the frame and the control word (CW) (Figure 3.5) have not been implemented to the full extent or employed, but the software possesses the necessary framework for it.

3.3.3 Implementation of the Interleaver

The software implementation is done according to Figure 3.6. All scrambling and convolutional interleaving is done in a separate class *Decode/Encode*. Scrambling and descrambling as well as interleaving has been tested to produce the same original data. For test and research purposes both functions can be switched off, so that the channel coding can be simulated without the scrambler and interleaver. This class encompasses the calls to the Reed-Solomon codec; additional erasure flags are always carried with the data throughout the decoder.

3.3.4 Cross Interleave Reed-Solomon Decoder Schemes

The algebra for decoding Reed-Solomon codes is based on Galois field arithmetic. All necessary operations on Galois field elements have been implemented. The author implemented his own version of a general Reed-Solomon decoder. Details of the algorithm

including the theoretical aspects are outlined in Appendix A.

The complete compact disc decoder encompasses two Reed-Solomon codes, C_1 and C_2, combined with three interleaver units (refer to 3.6). The Reed-Solomon codes are (32,28) and (28,24) decoders over $GF(2^8)$, respectively. Thus each of them is able to do two-symbol error correction as well as four-symbol erasure correction.

Generally, the first stage C_1 of the combination aims at correcting single random errors and detecting burst errors. Detected errors are passed on via erasure flags to the second stage C_2. If the first stage C_1 detects more than 1 single error, it usually outputs 28 erasure flags. Thus there is a choice in what shall be corrected by the first stage and what shall be done by the second error correction stage. The compact disc standard does not impose any particular decoder algorithms, that is why each manufacturer implements his own decoder strategy.

Error decoding schemes implemented in software and used for simulations are outlined in Section 2.4 in Table 2.1 and 2.2.

3.3.5 Sub-code Channel and Synchronisation Patterns

The encoded data format handled by the software does not include the control words at the beginning of the frame (Figure 3.6) nor does it contain the 24 frame synchronisation bits. The control words are not part of the error correction process, thus they are stripped off after EFM-demodulation and before error correction takes place. Storing information using control words is not discussed here.

Synchronisation bits at the beginning of each frame serve as mark for the electronics to recognise the start of a new frame. Destroying this information means the frame synchronisation is lost and, depending on the electronics, the whole frame is lost for

data recovery. This case is discussed as a burst error of one frame length.

3.3.6 Random Number Generator

Since many parts of the simulation involve using random numbers, a reasonable random number generator has been implemented.

Following the discussion in Numerical Recipes [43], it is not advisable to rely on the system-supplied library routine *rand()*. Instead the *ran1()* function, presented in [43] has been employed, unless stated otherwise. It is known to pass all statistical tests, unless the number of calls exceed, say 10^8 (which is not the case in this work).

Transformation methods are used to obtain exponential and Gaussian deviates according to [43]. The necessary functions are included in *Random.cc* and further explanation is given in Chapter 5.4.

3.3.7 Control of the Software

Apart from only providing an encoding and decoding machine, the software uses these core modules to evaluate certain test conditions. Control over these modules is granted through command line switches; the software program does not employ a graphical interface.

Subsequent Encoding and Decoding A stream of data is processed so that the output is channel encoded data according to the Standards. The encoded output is written to a file.

The corresponding decoding function uses the data stored in this file and decodes them back to the original data. Error introducing function can be operated on the file.

As an alternative, both encoding and decoding, can be executed subsequently in the same program execution. Error introducing is done in-between the encoding and decoding stages. Depending on the mode chosen, specific error patterns are generated and applied. Error patterns can be classified as systematic errors, random errors, bursts errors, etc. Each decoded frame is checked for the number of specific error conditions, as presented in 5.1.2. Error numbers are summed up and printed at the end of the simulation run for further evaluation. This method is used in Chapter 5 to investigate certain distributions of introduced errors.

Maximum Burst Length The maximal correctable burst length, as often listed in performance comparisons, can be evaluated for different decoder algorithms. The maximum correctable burst length for decoder II amounts to 12 frames. This is in accordance with published results [27], although this publication does not name the particular algorithm used. Older publications [44] give lower values (10 frames), which might be due to a non-optimum decoder algorithm.

Additionally one can define burst errors as a sequence of frames having wrong data only every second or third symbol in a frame and so forth or even showing certain wrong symbol patterns. For these kind of bursts, statistics can be generated as well.

Testing Bad Frame Patterns If subsequent frames are marked entirely bad, it is called a burst error. Such burst errors can be separated by a number of "good" frames. If bad and good sequences of frames are alternately ordered, a bar code-like pattern is obtained. The software is able to test the ability to correct such patterns.

Adjustable Error Rates A variety of other parameters can be passed onto the program. The error rate for introduced random errors must be specified when using

certain modes.

The form of the probability density function for burst errors is hard-coded into the program. Variations of error rates are possible through stretching the spectrum of the intermediate gaps by multiplying obtained length with a constant factor provided by a command line switch.

3.3.8 Computing Environment and Tests

The core of the program is the implementation of the Reed-Solomon decoder. It is crucial to test its correct working. A stand-alone version of the encoder and decoder was used to run a test of encoding and decoding procedures. The input data were random numbers. After decoding no difference was found between corresponding input and output data. As the decoder receives erasure flags at its input as well, one per symbol, combination of errors, correct symbols, erasure flags and no erasure flags were also tested. The decoder works correctly given the limits of the algebra Reed-Solomon decoding.

The same procedure was repeated for the whole channel encoding/decoding engine, including scrambling, interleaving, EFM- and NRZI conversion. The resulting output data stream revealed no differences from the input stream, which shows that the implementation of the decoding operations is the exact counterpart of the encoding operations.

Additionally separate tests for EFM conversion and NRZI conversion were carried out. Parts of the software like EFM, interleaving,... can be disabled, giving an easy method of running the test separately.

The software was run on two machines:

- A Pentium II based linux system with recent software packages. Two compilers were available here: The standard GNU Project C compiler (version 2.95) and the optimizing Intel(R) C++ Compiler for IA-32-based applications. The latter achieved a performance gain of 30%.

- A HP 9000/715 machine running HP-UX with HP's aCC++ compiler.

The implementation conformed to strict ANSI C++. The complete software listings are found in Appendix B.

Chapter 4

Results of Graphical Capture System

4.1 Methodology

The optical readout head hosts an array of four photo detectors, which pick up the main spot signal from the compact disc. This radio frequency (RF) signal is a weak signal consisting of the summed output of the four photodiodes (Figure 3.2) and is amplified by the player's circuitry. The compact disc player attempts to extract only the actual data content of the signal and discards all other modulations and distortions by using waveform shaping, extracting clock information and signal demodulation.

Surface contamination, local variations in the substrate and variations in the layer reflectivity stamp an additional modulation onto the signal. Thus the unprocessed signal carries a lot more information about the compact disc than is actually used by the CD player. Scratches can give an extra amount of reflection into the photodiodes, even increasing the signals amplitude.

Before the signal enters the waveshaping circuit, it is tapped off and digitally analysed by the author's software. The signal is digitised and stored in raw binary form onto the computer's hard drive, described in Section 3.1.5. Starting from here the data are processed using techniques adapted from image processing.

Sampling is performed as the compact disc spins under the readout head. The analog signal is thus continuously recorded and digitised for each revolution every T_s seconds. The resulting array will be called sampling matrix. The sampling matrix is built up of sampling points. Sampling points have a typical spatial distance of $14.6\mu m \times 1.6\mu m$ on the surface of the disc. The reflected energy $E(x, y)$ into the photo detectors, where x,y denotes the position on the reflective layer, can be expressed as

$$E(x, y) \propto r(x, y) t_V(x, y) \int_A I(x', y') t_s(x', y') dx' dy'.$$

$I(x, y)$ is the intensity of the laser beam on the surface and is modulated by a transmission function $t_s(x, y)$. The transmission on the surface depends on the absorbency of light by local surface imperfections and dirt.

$r(x, y)$ is a general expression for the reflectivity of the information (reflective) layer. More precisely $r(x, y)$ must be calculated considering the diffractive nature of the grooves ([35]). As the dimensions of the structures on the reflective layer to be resolved are not smaller than the spot size, it is not necessary to integrate over the spot, but leave it as a constant expression. Apart from that, structures which reside on the surface are much larger and need to be treated as integral.

t_V is an accumulated transmission coefficient for the absorbency of the substrate. Due to the fact that the substrate has a high transmission and local variations are small, this factor is not of interest for any further consideration.

Of importance therefore, is the integral over the surface spot area A. Because the area A for adjacent points (x, y) is overlapping, blurring occurs in the scanned image. The area A covers $0.5mm^2$. Considering the distance between sampling points, one single spot on the surface goes into about 22000 sampling points. The light amplitude distribution of the surface spot though is not constant but obeys a Gaussian curve, that is, the blurring is done with a Gaussian blurring operator.

Knowing the blurring operator a technique called *deblurring* can be applied to enhance image quality. Algorithms are readily available in the public domain, although the captured images in this work have not been processed in this regard.

When considering the resolution of the imaging system, two different situations must be distinguished: resolving structures on the surface of the disc and resolving structures on the reflective information layer. Due to the focusing of the laser beam, the resolution on the surface is much lower than on the reflective layer. Examples are given in Section 4.4.3.

4.2 Properties of the System

4.2.1 Resolution of the Scanning Process

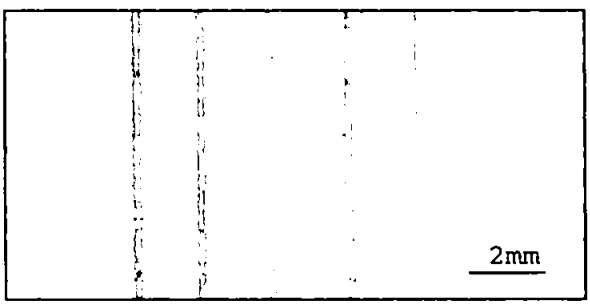


Figure 4.1: Photograph of radial scratches on compact disc.

It is obvious that there is a limit to resolve very fine structures on the surface. Fine structures are represented by very narrow lines or scratches. By assessing the signal response of the photodiode detecting the reflected laser, it is possible to get information about the real resolution of the system. Information about the signal deterioration can also be gained. Five radial scratches with a distance of $1.5mm$ have been placed on a CD. The scratches are not more than $20\mu m$ wide. Being made by a needle, they are still transparent and scatter the laser light. Figure 4.2 shows the signal response.

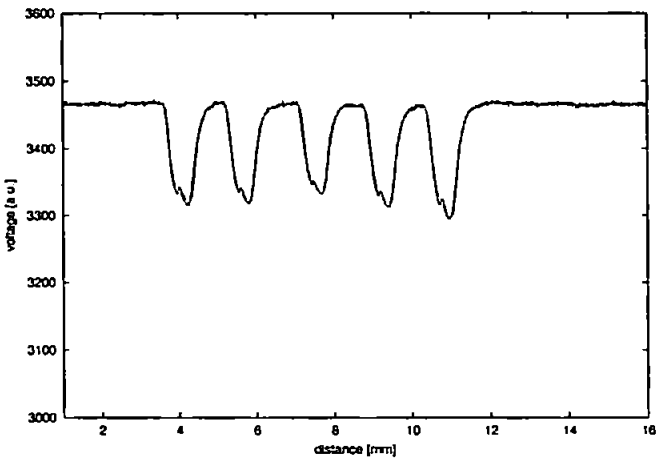


Figure 4.2: Laser-signal response to five radial scratches in compact disc.

As discussed in 3.1.2 the spot of the laser beam on the laser side has a diameter of $0.8mm$ [45]. The scratches placed on the laser side, as shown in the photograph in Figure 4.1, affect the laser beam only partly because of the non-focussed laser beam. The response signal shows that the scratches are widened to $1mm$ on the length scale on the surface of the disc.

In the following, resolution is defined to be the distance where two neighboured points can be resolved separately. The response signal suggests a resolution of $0.8mm$, based on the fact that the scratches can be moved closer to a distance of $0.8mm$ without not being able to separate them. This agrees with a spot size of the laser

beam of $0.8mm$ and is applicable to both directions.

It should be noted though that single points on the read-out side of much smaller dimensions can be detected, since they cause a distortion in the signal. It was observed that scratches and defects up to a few tens of a micrometer wide degrade the signal, which in turn can be evaluated by the software.

4.2.2 Signal-to-Noise Ratio of the Read-Out Signal

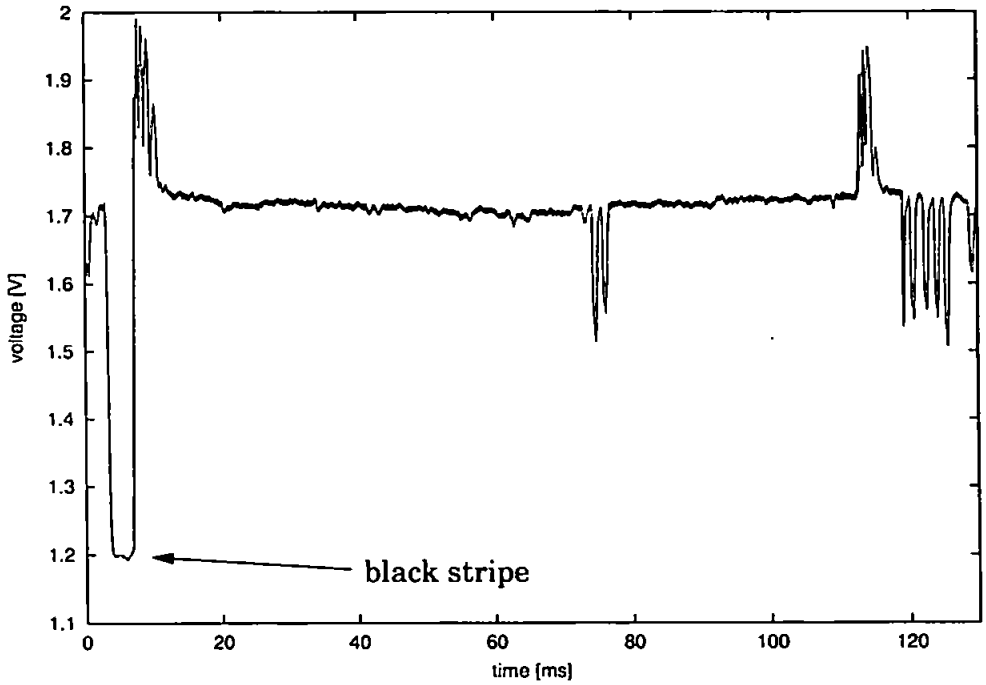


Figure 4.3: Voltage drop due to a black stripe on surface. The other peaks signify surface scratches.

The signal-to-noise (SNR) ratio is influenced by many factors. It is dependent on the type of CD-device, the ADC and the general experimental setup. The ADC is a 12 bit converter resulting in 4096 discrete voltage levels with a programmable input gain, adjusted to $\pm 2.5V$ for this application. Assuming a voltage of about 1.2V in the non-reflective case (black stripe) and a maximum of 2.5V with a high frequency noise

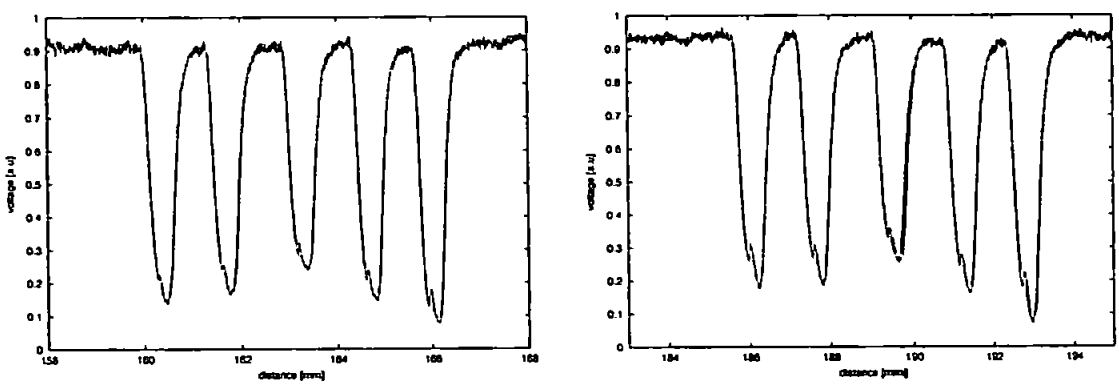
level of about 0.01V, the effective number of grey levels calculates to 106.

The original data channel of the signal is hidden in the high frequency noise, which is not considered as valuable information, since caused by the reflection of the pits and lands, the spatial resolution capacity of the system is not high enough to match this noise with any location on the surface.

Figure 4.3 gives the signal of one revolution. The drop is caused by a non-reflective black paper stripe. The minimum voltage is 1.2V. The peak-to-peak signal-to-noise ratio of the capturing system is around 30dB.

Remarkable is a short raise of the signal after the drop. It is caused by the servo focus control, which tries to recalibrate to the reflective layer. This effect can be seen in many surface captures, if the blemish is big enough to cause the servo control to lose the focus. In captured images it appears as a shadow after the surface distortion in the read-out direction.

4.2.3 Using Different Compact Disc Players



a) b)

Figure 4.4: Signal response of two different players to scratches on the compact disc. a) Toshiba XM3301B, b) Toshiba XM4101B

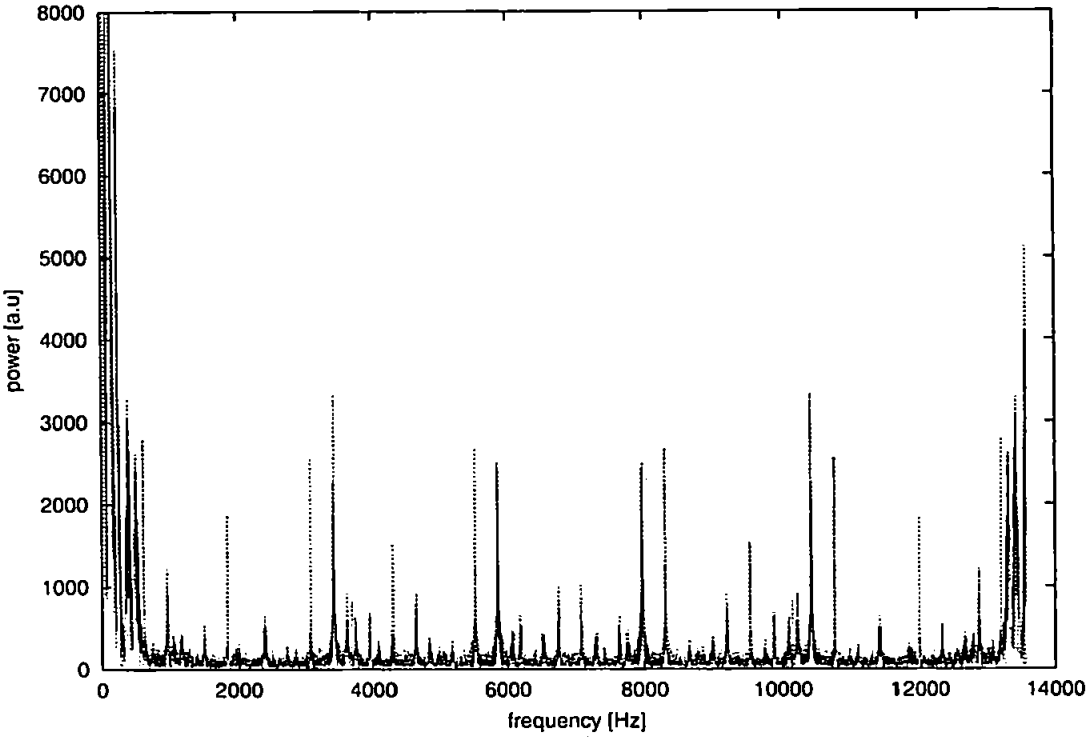


Figure 4.5: Fourier transform of data in Figure 4.4a (XM3301B).

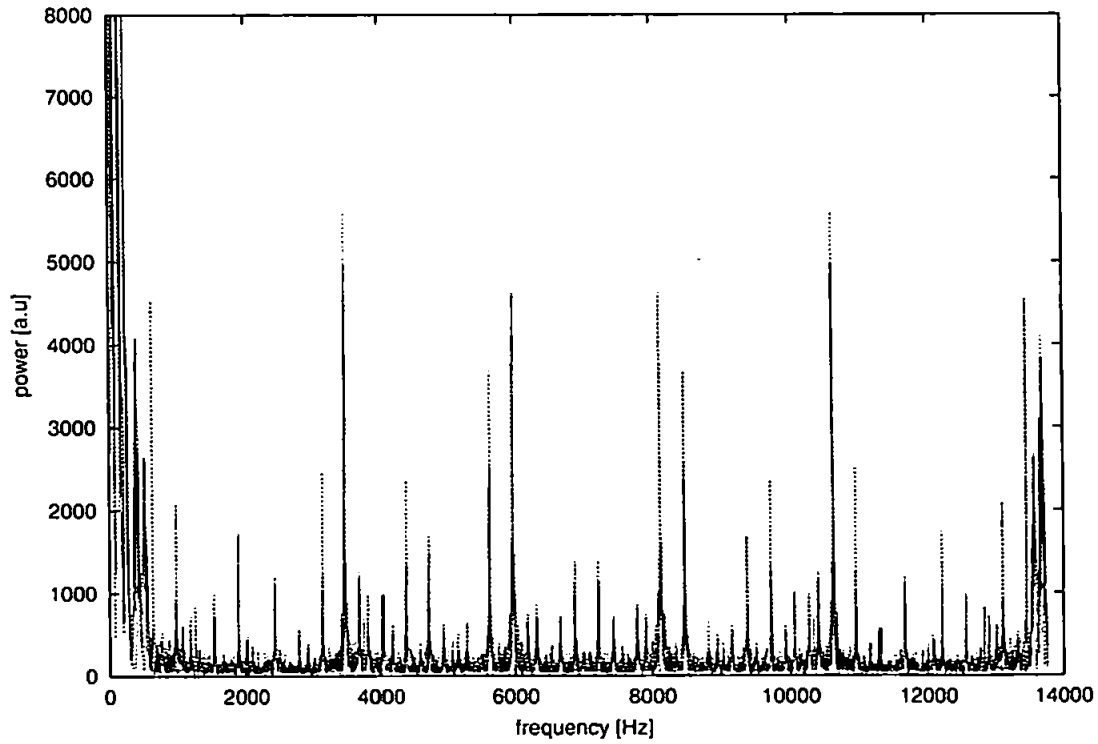


Figure 4.6: Fourier transform of data in Figure 4.4b (XM4101B).

The tests were carried out using two different CD players. This was because the author wanted to ensure the technique to be independent of the players used.

The two compact disc player have been compared in their signal response and signal-to-noise ratio. Both devices (Toshiba XM3301B, XM4101B) do not reveal any major differences neither in the signal response nor in the signal-to-noise ratio.

The signal response to above mentioned scratches (paragraph 4.2.1) are shown in Figure 4.4a and 4.4b. The signal is basically the same except that 4.4b shows more noise in it.

The respective Fourier transform of 4.4 is calculated in Figure 4.5 and 4.6, which shows the correspondence of both compact disc players in terms of noise and signal behaviour.

To conclude it can be said that despite their differences in the electronic circuitry and read-out head, the two compact disc players show the same characteristics. It can be generalised that the image capturing technique is practicable with all compact disc players.

4.3 Data and Image Processing Software

Raw image data, captured by the data acquisition software (Section 3.1.5), is stored in binary format. For each sampling point two bytes are allocated, since the ADC digitises into 12 bit values.

In order to further evaluate the raw data, an additional software program convert these data into picture file formats. This is carried out by scaling down the raw image data, so that it can be shown as an image. A zooming function is included to extract features of interest from the image. The zooming can be done successively, at each

stage the data can be converted to a picture file format and saved onto the hard disc. The file format used for saving these pictures is the *portable pixmap file format* (ppm).

Obtaining scaled-down versions of the raw data is desirable, because a captured surface image require up to several hundred megabytes of information. Normal image processing programs are not able to handle this, the author's program handle raw image data files of arbitrary size.

The software first calculates the scaling factor needed to visualise the image on the screen. Down-scaling is done by averaging over rectangular blocks of pixels. Colours are allocated by defining a colour map. The raw data values serve as an index to this colour map. Different colour maps can be applied to increase the visibility of interesting features. Colour mapping is not essential, because a grey level image is sufficient, but it poses a better visibility to the viewer.

A zoom function is incorporated into the software. Zooming can be done in batch mode, given a rectangular area to zoom in or interactively by using the mouse cursor. With the cursor a rectangular area of interest is defined and zoomed into.

At each stage the image can be saved as a ppm-file with scaling factor, thus allowing additional image processing software and filters to be used.

Batch mode allows extraction of horizontal lines as a graph, showing the deterioration of the captured signal in each track.

The compact disc's polar coordinates are remapped into rectangular coordinates. Each track of the compact disc is displayed horizontally, the y-coordinate denotes the number of the track. Thus a rectangular picture of the circular surface is formed.

Additionally the software is able to generate circular representations of the surface by displaying each track as a circle using polar coordinates. In this case interpolation

between points is carried out to fill the spaces between the sampling points, which leads to a slightly blurred appearance. In Figure 4.7 an overview picture of half of a compact disc's surface, taken with this method, is shown. Random scratches appear and are blurred out because of the mapping to polar coordinates. About 10000 revolutions of the disc have been recorded.

The program is command-line driven and uses the X-Window system for user interaction and visualisation of the data.

Figure 4.7: Captured overview of a disc surface with random scratches.

4.4 Interpretation of Results

4.4.1 Detecting Handwriting on the CD Surface

The surface of a compact disc is supposed to be clear and shiny without any distortions. The well-engineered CD system though allows a trade off of a certain amount of error correction ability for additional purposes. A first impression of the sensitivity of detecting watermarks on the surface is obtained by resolving hand-writing on a CD's surface.

For demonstration purposes the letters "KAY" (author's name) and "CRIST" (name of research group) has been written with a ball-pen. Figure 4.8 shows the captured image after image processing to enhance quality together with a scale bar. No adverse effect on the playability is noticed; CIRC errors are corrected according to the SCSI

The lines are displayed thicker than the original ones on the surface. This is due to the blurring effect of the unfocused laser spot on the surface even though the contrast is high enough to utilise optical character recognition (OCR) software to automatically identify the text.

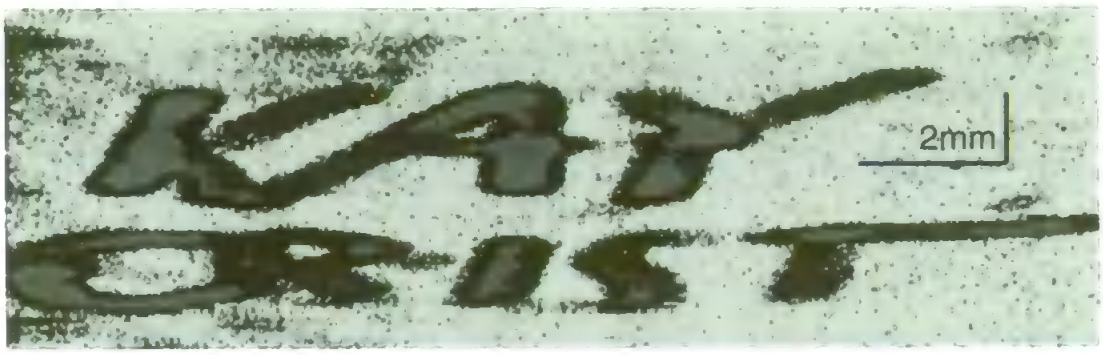


Figure 4.8: Image capture of the words "KAY" and "CRIST" written on the surface of a compact disc with a ball-point pen.

4.4.2 Fingerprints on Compact Discs

Fingerprints on user-handled compact discs are normal. The CD system is engineered to reproduce the music (or data) without loss of quality. Even if a degradation of the read-out signal is caused, the bit-content can still be restored in certain limits. If this is not possible, the error correction jumps in to rectify symbols detected as wrong.

Fingerprints act as a test bed for embedding watermarks on the surface. Fingerprints possess a very fine structure, are transparent so that the underlying data can be read, and are unique in their appearance. Due to their light absorbing nature the laser signal's intensity is reduced. The same effect can be obtained by modulating the transparency of the substrate, thus impressing transparent watermarks. Because the technical possibilities were limited this couldn't be achieved. Instead the author captured transparent fingerprints, almost invisible to the naked eye, on a compact disc's surface with this method. The fingerprints are impressed using very little red coloured ink, so that the fingerprint consists mainly of grease and a little ink. The software was optimised for this special task.

Figure 4.9: Captured fingerprint on the surface of a disc. The image was processed using an emboss filter.

The evaluation of the captured data shows clearly the ability to visualise faint

fingerprints on compact discs with great detail. It illustrates how small artefacts on the surface can be captured. An examination on the signal shows that an average 5% – 6% signal intensity is lost.

Disadvantageous is the occurrence of blurring because of a large surface laser spot. Structures of the fingerprint are smeared out so that special recognition software would be needed to identify them. This is a problem common to all surface capturing applications and can be refined by using image deblurring techniques.

Figure 4.9 gives an enhanced capture of a thumb print. Several image filter operations, including colour transformations, embossing, blurring were used to elevate important details.

The fingerprints must be faint enough to allow an error-free read-out of the compact disc. Evaluation of the SCSI error logs show that CIRC errors in this test were corrected. Problems were due to servo failure which is more likely to interrupt the playing. The control software handled this by rereading the specific sector again.

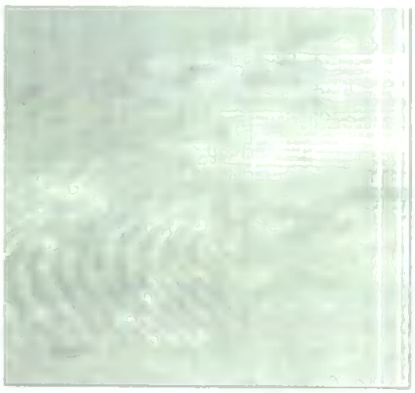


Figure 4.10: A second captured fingerprint with different filter operations applied (contrast stretching).

Reproducible results can be achieved as seen in Figure 4.10. A second fingerprint is captured, this time using different filter operations. Further image enhancing can be

obtained with specialised algorithms for fingerprint recognition. Filters applied include a non-linear filter, colourmap distortion, greyscale-converting, brightness and contrast adjustments.

4.4.3 Watermarks and their Capturing on Compact Discs

Future versions of DRM systems will involve watermarked content and medium along with redesigned devices which look for the watermark [46]. Control over digital content is ensured.

The compact disc medium possesses a number of possibilities in order to watermark it. As it was discussed in the previous paragraph modulating the intensity of the detected laser signal is one way. This can be achieved by local reflectivity variations of the reflected layer as well through absorbency variations of the compact disc's substrate and surface.

As long as the variations in the laser's intensity are small enough, the digital content will be finely reproduced. Even if it leads to short disruptions in the laser signal, the error correction would restore the original data (although with a loss of data redundancy for further error correction capability).

Watermarking of a compact disc in this sense is discussed in this Chapter. The process of impressing watermarks and the detection of them is successfully carried out.

Watermarking is implemented by mechanically puncturing either the surface or the reflective layer. This results in degraded laser signals due to modulated changes in reflectivity of the reflective information layer or due to additional laser light scattering on the surface, which the detection system (capture system) is able to process and to evaluate.

Both methods, the surface puncturing and the information layer puncturing, were found to work. Each one has their own characteristics. On the surface the resolution is lower than on the reflective information layer. This in turn has effects on the maximum density of watermark information. Furthermore the read-out technique has to be optimised for either method, even if both could be employed together.

The mechanical markings are about $50\mu m - 100\mu m$ in diameter and almost of round shape. Other introduction methods are feasible, an obvious method is by means of a strong laser burning deformations in particular positions. Single points can form symbols or code patterns which represents machine-readable information (Graphically Punctured Code).

Symbols on the Read-out Side

The front side surface was marked utilising a needle. A combination of up to 10 dot marks were placed on a square area of about $0.5mm \times 0.5mm$. It is found that the transparent marks have no adverse effect on playability, error correction is carried out successfully. No track loss appears either.

Since the blurring effect on the read-out side is very distinctive, the sharp dots are washed out and enlarged. This causes overlapping between adjacent dots and small structures cannot be resolved. In Figure 4.11 a three dot structure is shown. Each of these points are about $100\mu m$ in diameter on the surface. On the picture they are enlarged by a factor of 10.

Detecting is thus not a problem, the contrast is high enough to reproducibly obtain the coordinates of these points.

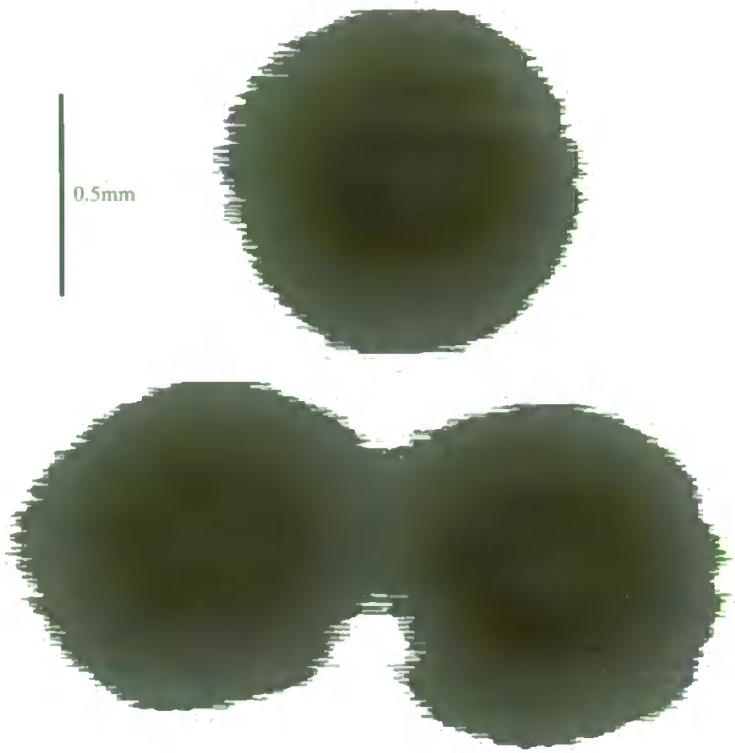


Figure 4.11: Captured surface image of three punctured distortions, compact disc plays without errors. The true size of the distortions is about ten times smaller.

Symbols on the Reflective Layer

The reflective inner layer is covered only by a very thin protective layer towards the label side. This makes it possible to puncture the reflective layer with a fine needle on the under side of the compact disc. Small deformations in the size of $50\mu m$ covering about 30 tracks on the disc are then obtained. The defects are imaged in the highest possible resolution due to a sharp focussed laser spot. The image is composed of points in the sampling matrix ($14.6\mu m \times 1.6\mu m$) with a size of $1.7\mu m$. No blurring occurs, because adjacent points do not overlap.

Figure 4.12 shows a group of four punctured dots imaged by the image capturing technique. The differences in colour intensity denotes the power of the reflected laser as received by the photodiodes. The image is not free of artefacts. While trying to

recalibrate, the focus control servo causes "shadows" after the blemish. Compared to punctured defects on the read-out side, these defects cause much more harm to the servo systems. By extending the size of the defects by only a few more microns, the CD is rendered unplayable. Thus the size of the deformations has to be as small as possible.

Figure 4.12: Four punctured distortions in the reflective layer, imaged by the author's capturing technique. The size of the small dot is about $100\mu m$.

Additionally photographs were taken with a scanning electron microscope (SEM) and an optical microscope. Figure 4.14 shows the scanning electron microscope picture and Figure 4.13 the optical microscope picture. Comparing the three methods the author's image capture technique is verified. Even the small reflective layer distortions around the main points as seen in the optical picture show up as artefacts on the image capture picture. Compared to the optical microscope picture the image capture technique reveals more three dimensional information. It must be noted that the electron microscope pictures the distortion from the under side (label side) of the compact disc, whereas with the other two methods the photographs are taken from the read-out side.

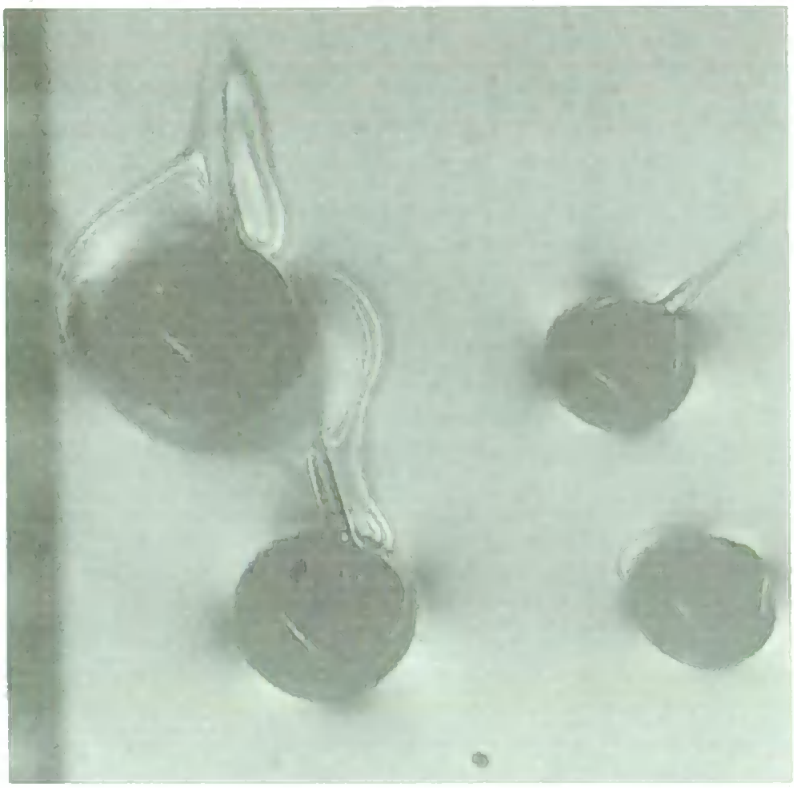


Figure 4.13: Photograph of this distortions taken by an optical microscope. Compare to Figure 4.12.

Figure 4.14: Photograph of the same four distortions taken by a scanning electron microscope. Compare to Figure 4.12.

To conclude it can be said that the author's image capture system of the reflective layer is comparatively equal to that of other methods in terms of spatial resolution. Applications like quality assurance of compact disc production come quickly into mind. The cost of this system is by far lower than all other methods. Image capturing for watermarking purposes will be discussed in the next paragraph.

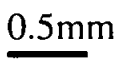


Figure 4.15: A group of punctured holes in the reflective layer, captured during the playing of the compact disc.

Finally Figure 4.15 shows a group of punctured holes in the information layer. The playback of the compact disc is not interrupted during playing over these blemishes. Figure 4.16 shows one of these distortions in the maximum possible resolution. A

Figure 4.16: One punctured hole in the reflective layer, captured with maximum resolution during the playback of the compact disc.

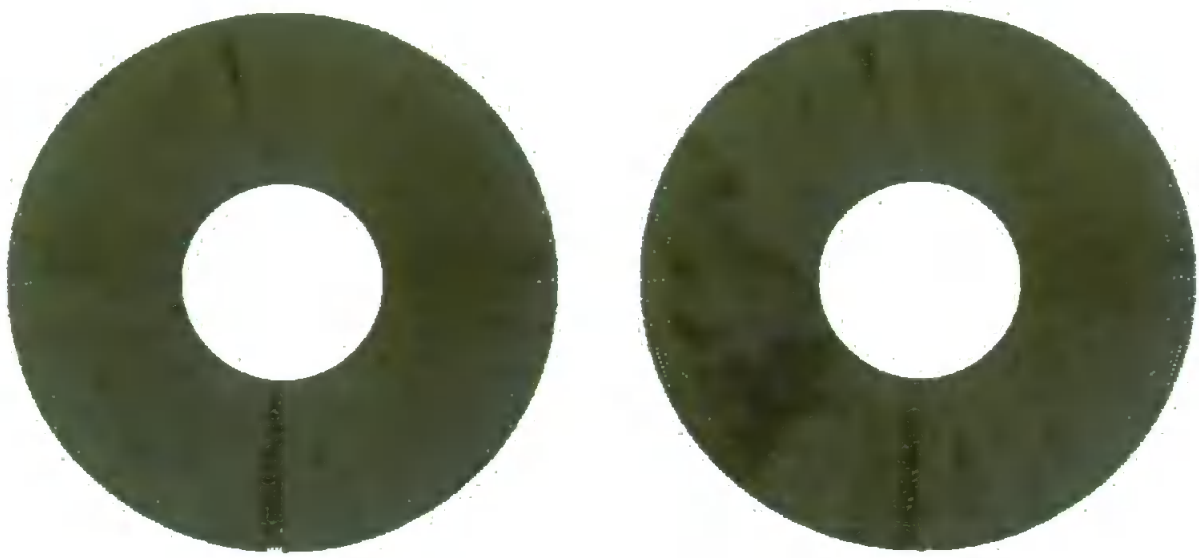
scale bar is included. The shadow in x-direction caused by refocusing is clearly visible. Different colours mark a different depth. The resolution, being in the micrometer scale, is surprising, considering the low-cost mechanism of this system.

4.5 Further Applications

4.5.1 Random User-Handling and Dirt

User-handling of the CD leaves fingerprints, scratches, and other dirt on the surface. These natural blemishes are detected and well visualised by the proposed image capture technique.

The surface of a heavily scratched CD cleaned with white spirit is shown in Figure 4.17a). The darker the colour, the more the signal is deteriorated. In comparison in Figure 4.17b) the surface is captured after exposure to extensive user-handling. Even if the signal loss is about 25 % compared to the lowest signal gained with a black stripe, the CD device plays the music without any audible errors. The capturing was done with 4000 rotations of the CD starting 4 minutes after the beginning of the CD and took 9 minutes.



a) b)

Figure 4.17: Two captures of the same disc. a) cleaned disc b) after touching. The shaded areas mark lower reflection caused by grease and dust.

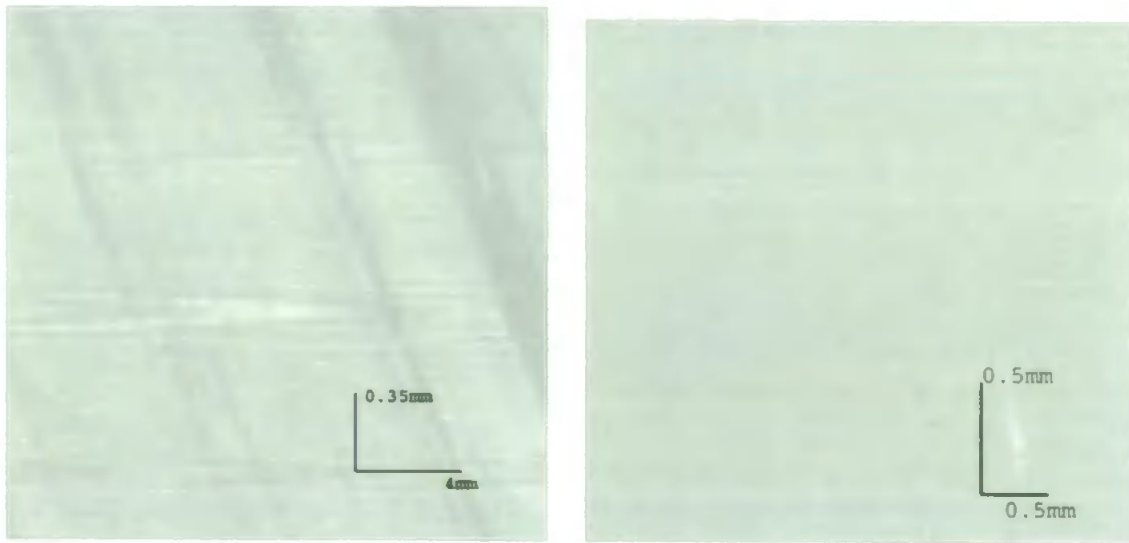
4.5.2 Moiré-Pattern

The captured image of one of the compact disc examined reveals a moiré-like pattern. The disc is badly manufactured; the same disc shows defects in the reflective layer like those discussed in paragraph 4.5.3.

Figure 4.18a shows this moiré pattern together with perpendicular stripes and scratches which are actually remains after cleaning ink off the CD. It is believed that the structures are related to an inhomogeneous polysubstrate layer. Figure 4.18b shows an image of a disc without manufacturing artefacts.

Figure 4.19 shows a demonstration of an injection fault during the disc production. This photograph is taken by a commercial compact disc manufacturing monitoring device, marketed by Basler AG, Germany. The similarity of both images suggests the same cause.

The moirè pattern repeats about 40 times over the surface, while the period remains



a) b)

Figure 4.18: a) Moiré-pattern in the substrate detected on a badly manufactured disc b) a clean, good disc without moiré-pattern

Figure 4.19: Demonstrative example of an injection artefact in badly manufactured compact discs (Basler AG, Germany).

almost constant. There is only a small difference in the period between the inner parts and the outer parts of the surface of less than 5%.

The same figure gives another interesting feature of the capturing process. A rounded, irregular line in the bottom part of the image can be seen. This is caused by a dried stain of white spirit mixed with ink from a former cleaning process. Despite all these variations in the laser signal intensity the disc plays well without causing any audible error.

4.5.3 Detecting Manufacturing Defects during Read-out

Optical media manufacture consists of various production steps such as injection molding, sputtering, protective coating and so on. These process steps imply possible defect sources which can compromise the playability of the medium.

While using and examining a number of compact discs, one has been found to have substantial defects in the reflective layer. With a strong backlight these appear as tiny little holes. The diameter of these varies typically from $10\mu m$ to $50\mu m$. There are tens of them per square inch dependent on the region. The cause is obviously a bad production lot, because the same disc shows a moiré-pattern as well as discussed in 4.5.2.

As shown in the previous Chapter the capturing system maps the reflective layer of a compact disc accurately. Applying this method to a badly manufactured compact disc, the result shows these artefacts as well. An image was captured of this disc; the overview is presented in Figure 4.20. This picture shows all artefacts well, a software emboss filter was used to enhance small details.

The read-out is done in the x-direction. Perpendicular stripes are remains from

wiping the surface in the polar direction. The small spots represent the defects in the reflective layer. A moiré-pattern is visible, it is believed that these are structural artefacts of the polysubstrate layer, since it is only discovered on this disc. The image shows the increasing length of the tracks in the y-direction.

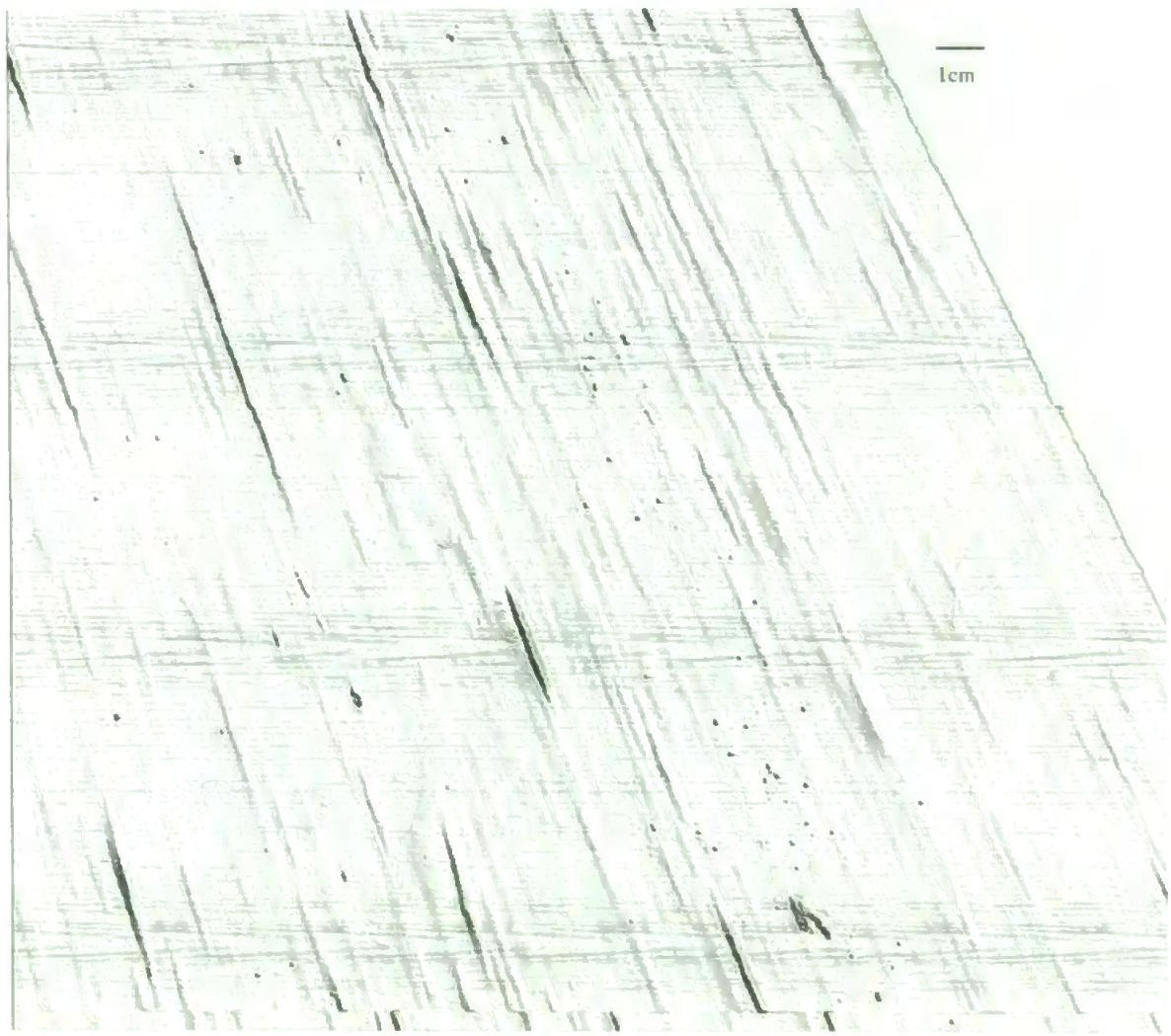


Figure 4.20: A captured overview picture of a badly manufactured disc with lots of defects seen as small dots.

The compact discs plays without CIRC errors being observed.

Pictures have been taken with an optical microscope to prove the existence of said blemishes in the reflective layer. Figure 4.22 shows a part of the read-out surface, the black spots being the defects in the reflective layer. The wide stripe is drawn by a ball

Figure 4.21: A zoomed-in version of 4.20, showing distinctively the defects, moiré-pattern and a part of the boundary.

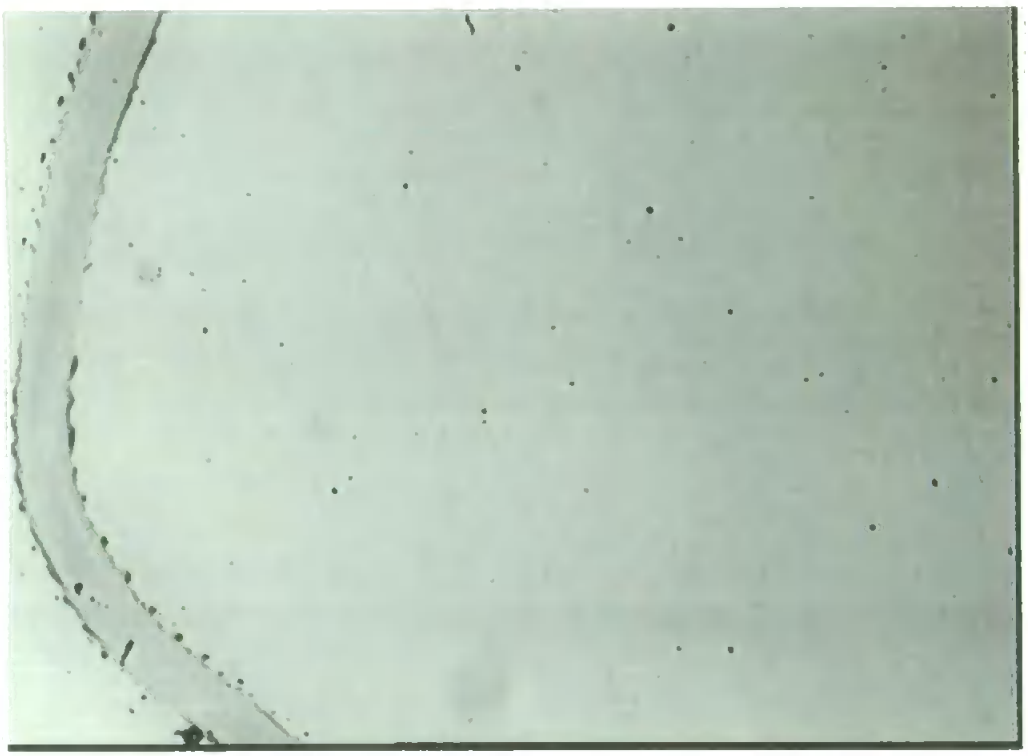


Figure 4.22: A photograph of the surface taken by a conventional optical microscope.

pen to mark the boundary of a region.

Figure 4.23 shows a captured image of a region surrounded by a ball pen line. This region is populated densely by reflective layer defects. Figure 4.22 shows a part of this region photographed by a conventional optical microscope.

Comparing the two figures 4.22 and 4.23 the difference between both methods becomes obvious. Whereas the microscopic image shows the details more accurately, Figure 4.23 reveals more details in general like moiré-pattern and blemishes on the surface.

It must be noted that the captured images are shrunken versions of the original captured image. The resolution is, as discussed in paragraph 4.4.3, $14.6\mu m \times 1.6\mu m$.

A badly produced compact disc with such defects has actually its own watermarks stamped in the reflective layer. Although the watermarks obviously have not been

Figure 4.23: Image capture of the disc with a circular boundary drawn by a ball-pen around reflective layer defects.

placed there on purpose, this specific damaged compact disc is possibly the only one having this characteristic clusters of defective spots in the information layer. Because it is believed that during manufacturing these spots have been generated randomly, the disc is identifiable uniquely even amongst other compact discs of the same production lot.

Thus we might say this compact disc is fingerprinted by the manufacturer. Detection and evaluation of the digital fingerprint has been done in this Chapter. The playability is not adversely affected, because the missing or destroyed data are restored by the error correction system. A small amount of the error correction's capacity is used up though.

Detecting digital fingerprints in this way can take too much time, because the whole disc must be scanned to get all information. It is therefore convenient to distribute them along a few tracks, as discussed in paragraph 4.4.3. Because errors in the digital data content caused by the introduction of such watermarks must not only be correctable but also present the best correctibility, regarding also the natural background noise

(dirt, fingerprints), to the error correction system, a thorough investigation about the possible locations, regarding their positions relative to the respective frame beginning, must be carried out.

The next Chapter raises this point and continues the discussion by the investigation and evaluation of watermark introduction, backed up by computer simulations.

4.6 Conclusions and Applicability for Watermarking

The image capture technique works well for getting high quality images of a compact disc's surface or the underlying reflective layer. Apart from this it offers a reliable way of examining the compact disc for imprinted mechanical watermarks, both on the read-out surface and on the reflective layer. This Chapter focussed on imprinting, detection and evaluation of introduced blemishes which can serve as a digital fingerprint of a single compact disc. The control software was able to read back the positions of introduced blemishes. The effect on the error correction was checked by evaluating SCSI error logs, which give a reasonable statement about the occurrence of CIRC errors. The blemishes were introduced in such a way not to jeopardise the playing of the compact disc, although the reasons for choosing the exact positions are not comprehensible at this point. A discussion about optimal locations will be given in Chapter 5.

The interleaving and error correction system gives rise to the assumption, that certain positions and certain patterns are more correctable than others, thus it is possible to optimise the information-to-output ratio for watermarks.

Simulating the channel coding for compact disc will help to judge error patterns

and certain positions within a frame or successive frames on their correctibility. The experimental trials are replaced by a more scientific and reliable approach. The aim is to achieve the highest information-to-output-error-rate ratio. Also, a requirement is that possible watermarks are read out as quickly as possible. This means they must be arranged in such a way that during one or two revolutions of the disc, all watermarking information has been read. Thus a two dimensional code must be considered. It was shown that mechanically puncturing the reflective layer is enough to obtain reasonable sized and detectable watermarks. Instead of using a mechanical device, a strong laser would be more capable of precisely marking positions. By arranging the markings in a defined way, a code can be formed. This proposed *Graphically Punctured Code* will be derived in Chapter 5.

Chapter 5

Watermarking and Punctured Code Simulations

5.1 Definitions and Background

5.1.1 Error Concealment

When the capability of the error corrector is exceeded but errors are detected, the uncorrected but flagged samples should be concealed if audio data are processed. Concealment reduces the noise resulting from uncorrected samples. Various techniques are available and are actually implemented in compact disc players.

muting: The value of the erroneous sample is set to zero.

previous value-holding: Zero order interpolation.

first order interpolation: Mean-value interpolation. The erroneous sample is replaced by a level midway between the previous sample and the following sample.

n-th order interpolation: Uses a polynomial approximation of n-th order.

According to the data sheet of a modern single chip decoder (Philips SAA7325) [5], concealment with more than one consecutive non-correctable sample is achieved by holding the last good sample and performing a one-sample linear interpolation before the next good sample. In general all CD players use two independent interpolators for the left and right channel.

Many publications ([1, 47, 28, 45]) use the probability of encountering an interpolated sample P_{ip} or the probability for audible clicks P_{click} in order to give a rough impression of the capabilities of a certain error correction system depending on the input symbol error rate P_{symbol}.

This convention is convenient, since these terms address the issues of the listener. Error rates in this work are calculated in terms of P_{ip} and P_{click} amongst others. Additionally one can consult the definitions for $P_{00}, P_{01}, P_{10}, P_{11}$ given in Chapter 2.6. These are also computed by the software and used in some cases. They relate to previously mentioned ones in the following way [1]:

$$P_{ip} := P_{interpolation} = (P_{11} + P_{01}) \cdot (2 - P_{11} - P_{01})$$

$$P_{click} = P_{10} \cdot (2 - P_{10})$$

These equations take into account that each mono-audio-sample consists of two symbols. The click rate basically expresses the probability of a misdecoding plus a not-detected error. When encountering such a situation, the CD player generates an audible click in the loudspeakers, which should be avoided by any means. The interpolation rate refers to a first order interpolation of two erased symbols (mono-audio-sample).

5.1.2 Definitions of Error Rates

Performance of error correction systems can be expressed in an objective manner by specifying the click rate, the symbol error rate or the interpolation rate of the decoder strategy in its functional dependence on the input error rate. In general, error rates are measured in erroneous bits over total bits. This equivalents to the probability of encountering an erroneous bit and therefore the units are omitted in the following.

Due to some confusion in other publications in defining these terms, an exact definition of output error rates, as assumed by the software, is given in the following.

interpolation rate P_{ip}: An event is counted as "interpolation" if one or two symbols of the 16-bit one channel mono-audio-sample are marked as an erasure, independent if it is an error or not and surrounded by not erased symbols.

symbol error rate P_{sym}: Every symbol is compared with its original value. The not-matching symbols are counted to the symbol error rate, not considering if the symbols are erased or not.

sequence error rate P_{seq}: Sequences of wrong symbols preceded and followed by not erroneous symbols are counted.

click rate P_{click}: An event is counted as "click" if there are one or more symbols not erased but erroneous between either not erased symbols or not erroneous symbols. This is a measure for mis-detection and mis-correction by the CD player.

5.1.3 Limits of Computer Simulation

It should be noted that the resulting probability values can only be an approximation to the "real" values. The approximation will be better as the number of iterations

increases. This in turn means that for events with a relatively low probability, more iterations are necessary. Otherwise large measurement errors in terms of standard deviation have to be accepted.

Number of detected errors	1	10	40
deviation error with ±1 error	100%	10%	2.5%
Probability P_{symbol}	$1.66 \cdot 10^{-9}$	$1.66 \cdot 10^{-8}$	$6.7 \cdot 10^{-8}$

Table 5.1: Lowest error probability to detect when decoding $2.5 \cdot 10^7$ frames with 1,10 or 40 symbol errors occurring. One additional symbol error leads to the resulting standard error.

Decoding 25 million frames ($1.66 \cdot 10^8$ symbols), the lowest symbol error rate to detect is $r = 1.66 \cdot 10^{-9}$. If 10 errors are to be expected, the lowest rate amounts to $r = 1.66 \cdot 10^{-8}$. Assuming an error deviation of one symbol error per measurement, measuring an error rate of $r = 1.66 \cdot 10^{-8}$ (one symbol error) means a measurement error of 100%. Table 5.1 gives an overview.

Assuming a reasonable value for the deviation to be 2.5%, a minimum number of 40 symbol errors at least must be expected. Accepting no less than 40 errors is common practice.

In this work the highest number of frames iterated was 25 million. Consequently a minimum error rate of $P_{symbol} = 6.7 \cdot 10^{-8}$ could be detected at best, when accepting no less than 40 errors. The results were scanned for error counts less than 40, which were rejected. In some rare cases error counts of above 20 have been accepted which equals an uncertainty error of 5% maximum.

The wide range over several orders of magnitude of the error rates requires the same range in computing time. Comparing the output error rates of $P = 10^{-3}$ and $P = 10^{-6}$, the latter one takes a thousand times longer to compute. Thus computer simulations in this area are always supposed to have a bottom-limit due to the computing time

available. The software was optimised for speed therefore.

5.2 Conformance of Computer Simulation and Statistical Analysis

Results of the author's computer simulation have been matched against published error rates, obtained from statistical analysis of decoder strategies. In [1] such a calculation is presented. A memoryless channel is assumed and the interpolation rate is calculated using a statistical approach. The decoder algorithm used is number I.

A memoryless channel model describes the errors as occurring statistically independent from each other. It is a first approximation to a real channel, where the error distribution shows bursts of errors.

The author's software implements decoder I, along with a modern version, decoder II. In Figure 5.1 the straight line represents the interpolation rate calculated in [1]. Compared to the interpolation rate, obtained from the author's software, conformance is found.

5.3 Decoder Algorithms applied to Memoryless - Channels

Two decoder algorithms have been implemented. In order to find out which one performs better, they are both applied first to a non-bursty (memoryless) channel and in the next Section to a bursty channel.

Computer simulations leading to P_{xx} values and error rates as listed in Section 5.1.2

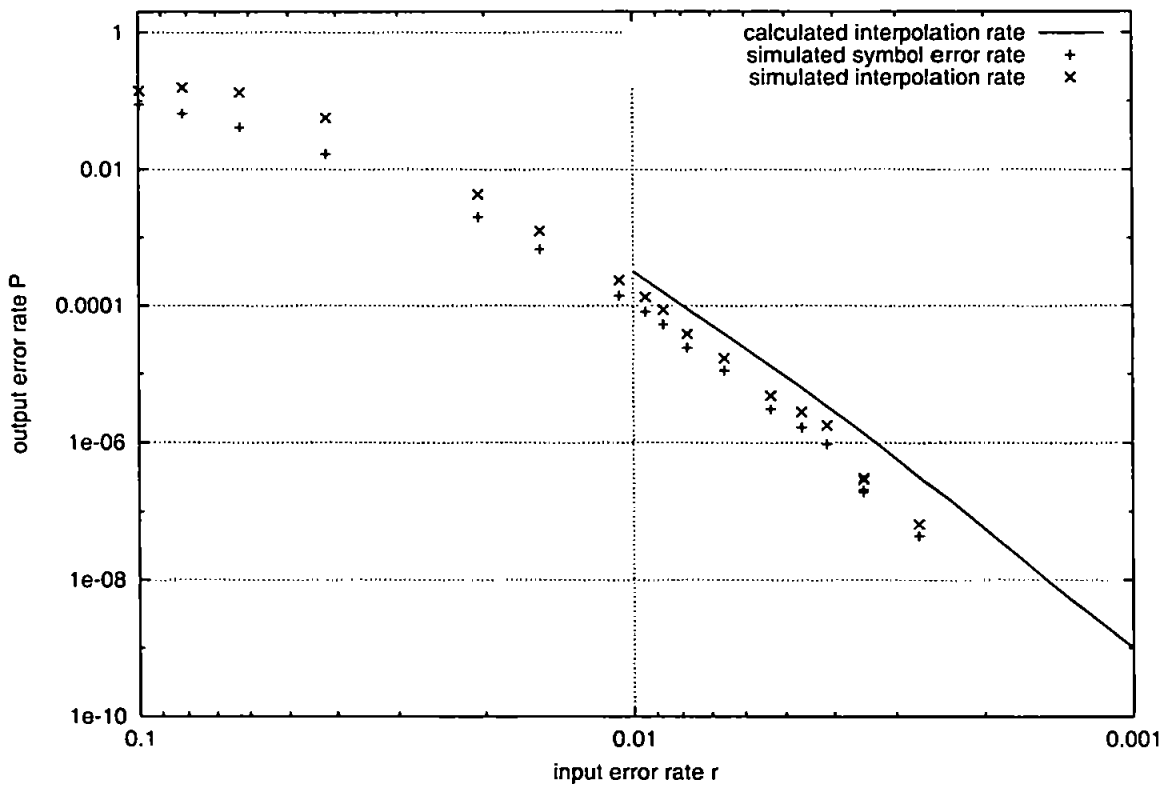


Figure 5.1: Diagram showing interpolation and symbol error rate (crosses) as obtained by the author's software. The straight line represents the interpolation rate, published in [1] as a result of statistical calculations.

for these two decoders have not been carried out before to the knowledge of the author.

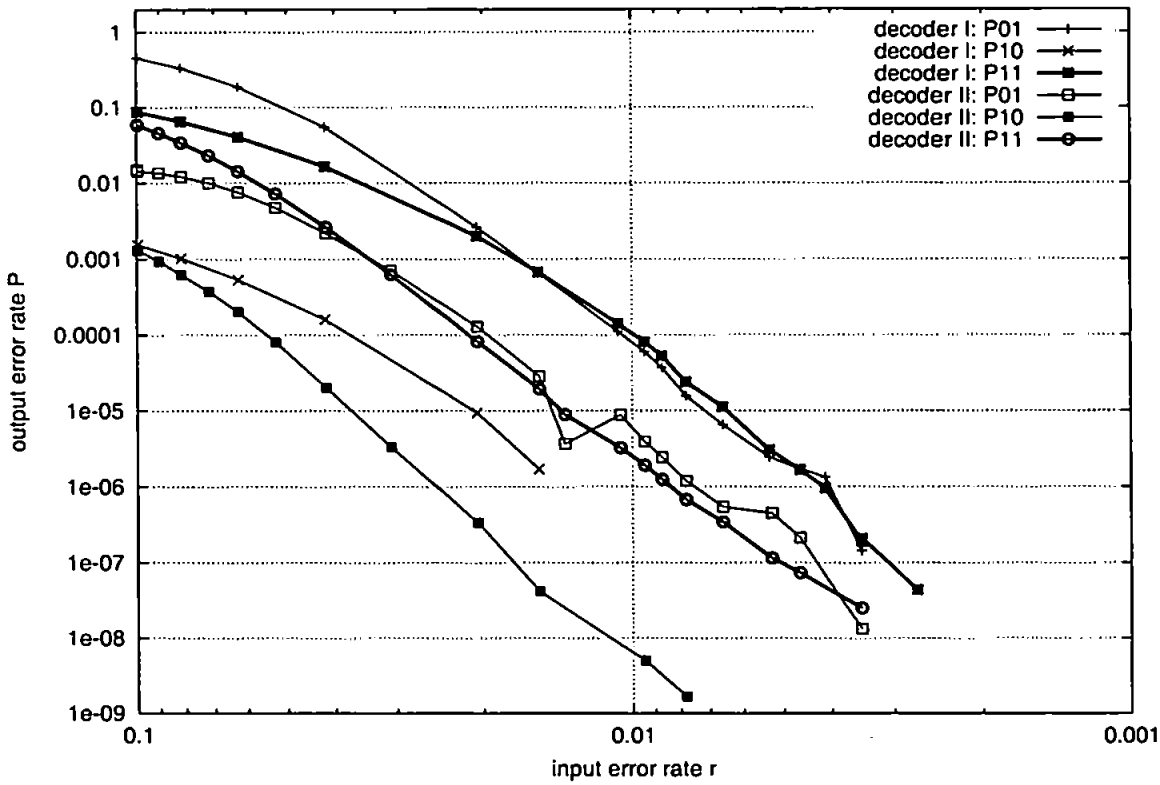


Figure 5.2: Decoder I and II compared in terms of P_{xx} values on a non-bursty channel. Decoder II shows better performance.

The results are shown in Figure 5.2, which shows the output error probabilities P_{xx}, and Figure 5.3, which shows the other error rates. Each diagram compares decoder I and II as presented in Table 2.1,2.2.

Only results with a number of errors greater than 20 have been considered. That means one error more or less in the output cause an error of maximal 5% in the output error rates.

It becomes clear that the author's implementation of decoder II has a higher performance in terms of error correction rate. At an input error rate of $r = 0.04$ the difference in P_{xx} is about one order of magnitude.

The next Section applies these two decoder algorithms additionally to a bursty

channel. The decoder with better correction capacity will be chosen for simulation of watermark sequences.

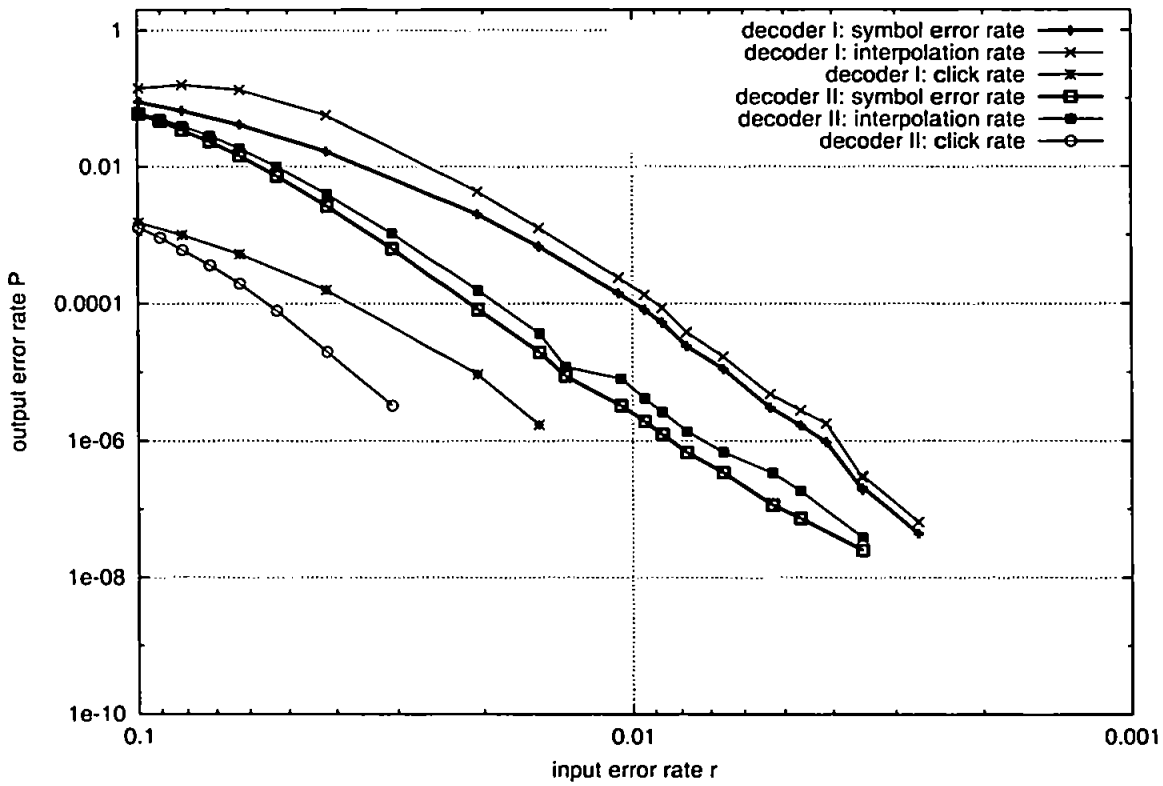


Figure 5.3: Decoder I and II compared in terms of interpolation, click and symbol error rate on a non-bursty channel. Decoder II shows better performance.

5.4 Decoding Burst Errors

5.4.1 Introduction

Optical storage media show not only random errors, but burst errors. Burst errors are defined as groups of erroneous symbols. The distribution of burst errors is obtained by error measurement equipment [32]. Burst errors result from heavily damaged discs, where the scrambling mechanism is not able to spread them apart. It can also be generated by introducing very few errors but at certain known positions. Burst errors

show different requirements for the decoding strategy. The decoding process must be able to cope in particular with these errors, because they are characteristic to the compact disc's readout process.

5.4.2 Reproduction of Error Burst Probabilities and Good Data Gap Probabilities in a Bursty Channel

It was the aim of this work to incorporate the presence of a certain probability distribution of burst errors in order to present a model close to reality. As it will be shown later, the characteristics of the background noise, in the form of a probability density function, influences the ability to correct certain types of error characteristically.

Various publications deal with the reproduction of error distributions in bursty channels. For the characterisation of the real communication channels there are several channel models available. One such model is the well-known Gilbert model [48]. It comprises only two states: the bad state "B" and the good state "G" with the respective probabilities to change or keep the state (Figure 5.4).

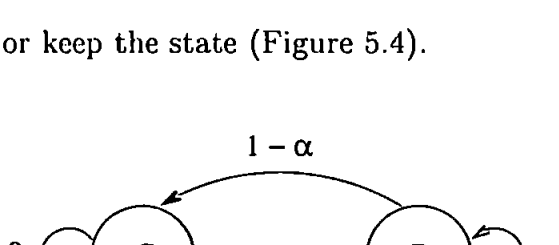


Figure 5.4: Gilbert model

However this model is proved not to reproduce measured error distributions exactly. A more precise model was proposed in [49]. Since a characterisation of the compact disc channel is not needed here, only the resulting probability distribution is important

in this work.

The author's simulation of bursty background errors is based on an error distribution published in [2] and experimentally determined in [31, 32].

This error distribution is emulated by a combination of random number deviates. The output of these represents the probability of the occurring of burst errors and matches the graph in [2]. Figure 5.5 plots the burst relative frequency f in a half-logarithmic scale against the burst length. The relative frequency of the error bursts is obtained by dividing the number of bursts of a given length by the total number of burst events. Thus the area under the curve is standardised to one.

Curve b is the same as presented in [2], emulated by specific random number generators. The generator is composed of three different deviates, namely one exponential for the steep slope at the beginning, one Gaussian for the peak at about burst length 30, and one last exponential for the low probability of encountering longer burst errors. They combine with different probability, so that they can give this characteristic curve. This describes the background errors of a "clean", not subjected to extensive user-handling, compact disc.

Simulations in this Chapter are done using curve c, which has a higher probability for bursts with a typical length of around 30 symbols. It is believed that this gives the typical characteristic of a user-handled disc with scratches on the surface. The probability of encountering a burst here is ten times higher than on the "clean" case.

For comparison curve a is the one given by simulating a memoryless channel, where no burst errors occur. The complete different shape is resulting in different error correction rates, even if the symbol error rate before correction is the same.

All three curves are standardised to their respective area. In order to get the ap-

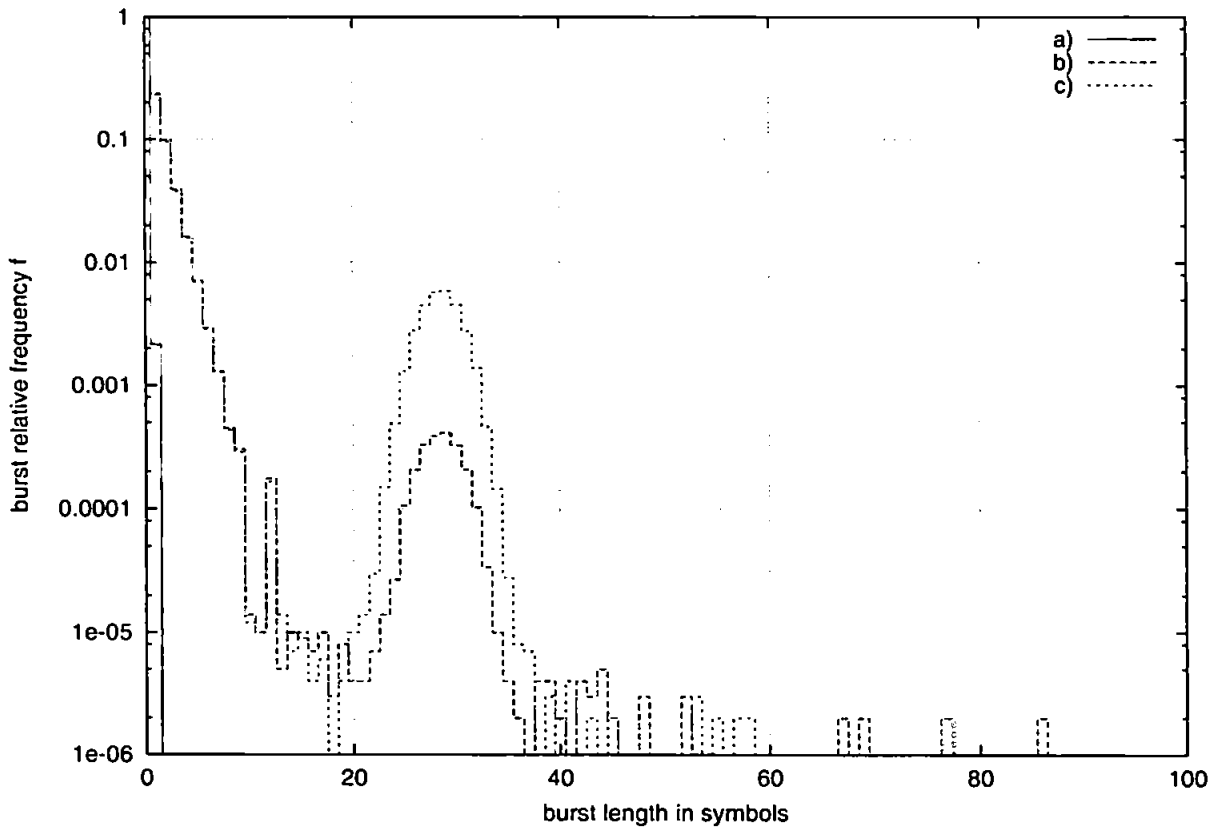


Figure 5.5: Different probability density functions used for simulations of bursts (area under slope standardised to one). a) bursts caused by a memoryless, non-bursty channel, b) bursts emulating measured distribution in [2] on a clean disc, c) bursts with a ten times higher probability assuming a scratched disc. Curve b and c is reproduced using a combination of random number deviates.

propriate error rates resulting from these error statistics, the good-data gap statistics in-between the error sequences have to be emulated. A similar procedure for reproducing gap statistics is carried out.

The gap statistics basically consists of a part of high probability short gaps and a bulk of gap lengths, having a constant, ten times lower probability relative to the short length gaps. The longest gap length is 10000 symbols.

Varying the input error rate with these two independent probability functions is done by scaling the gap distribution by a stretch factor n. The lower the input error rate is supposed to be, the higher the average gap length, obtained from the gap distribution, must be. It is believed that the characteristic of the error statistics does not change when dealing with a more heavily scratched disc, i.e. the slope of the burst density curve remains the same. Instead the intermediate gaps vary in their average length. This manifests in a stretching of the gap statistics in order to achieve different error rates.

Input symbol error rates ranging from $r = 0.0001$ to $r \approx 0.8$ have been obtained in this way. The relation between stretch factor n and obtained input error rate r is shown in Figure 5.6.

The following simulations have been carried out using the error distribution of Figure 5.5c. It is believed that this results in quantitatively and qualitatively more accurate results which come closer to real-world applications.

5.4.3 Decoder Algorithms Applied to Bursty Channels

Figure 5.7 presents the error correction capacity in terms of symbol error rate and interpolation rate for both decoding strategies I and II. It is evident that both of them

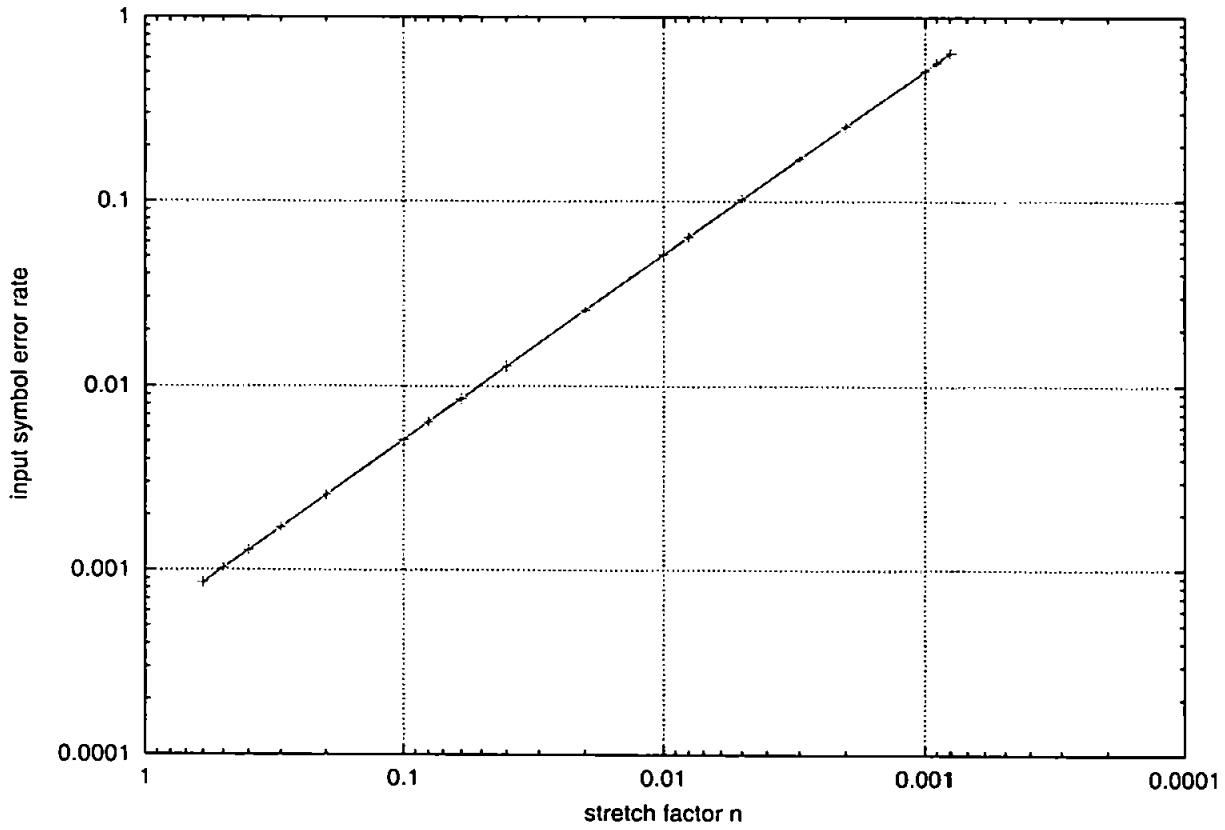


Figure 5.6: Relation between stretch factor of gap lengths n to obtained error rate r. The error distribution is constant while scaling the gap distribution with factor n to achieve different input symbol error rates.

can vary up to one order of magnitude from each other.

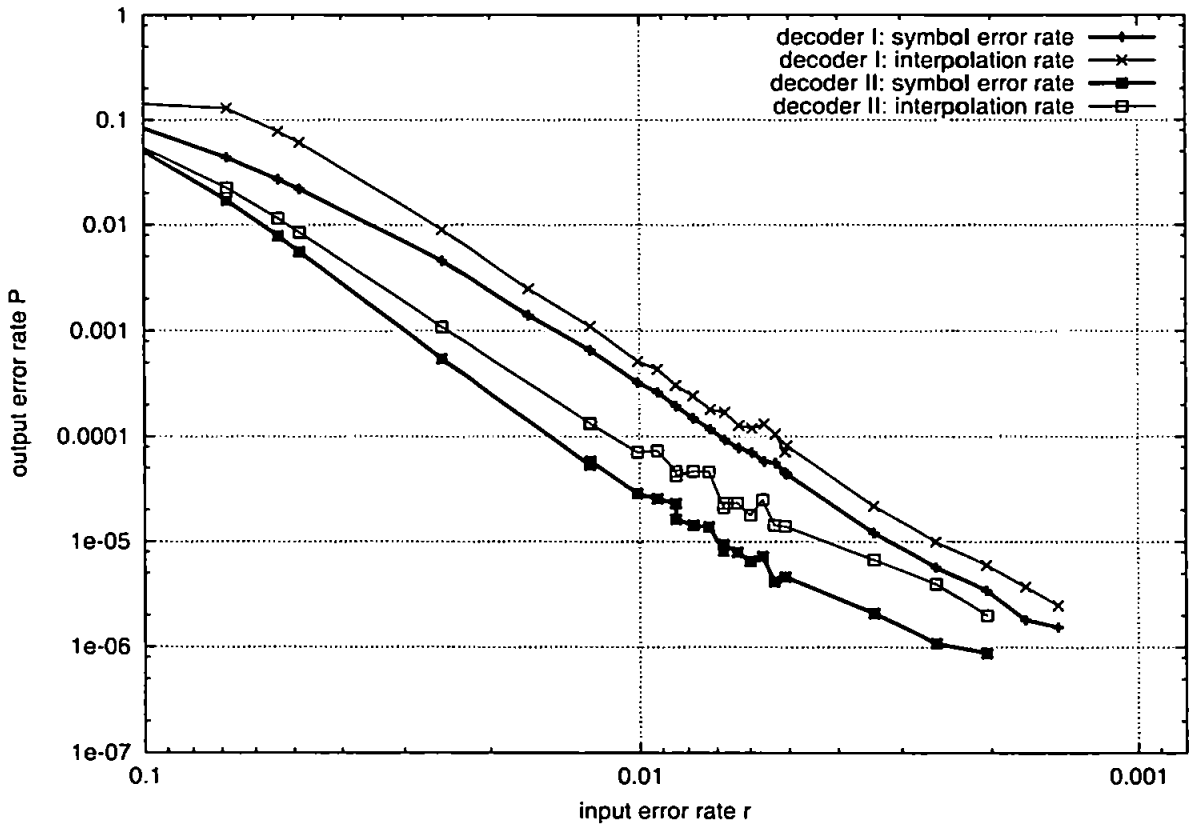


Figure 5.7: Decoder I and II compared in terms of symbol error rate and interpolation rate on a bursty channel. Decoder II shows the better correction performance.

An interesting feature can be observed at very high input error rates $r \approx 0.2$. The interpolation rate decreases substantially for decoder I. This might be due to very long sequences of erased symbols which actually do not count as interpolated sequences anymore (according to the author's definition) because long erased sequences are muted.

The parameters P_{xx} have been calculated as well. Due to a low number of P_{10} events, it was reasonable not to choose them for a comparison.

Interpolating to an input error rate of $r = 0.001$, the output symbol error rate becomes approximately $P_{symb,bursty} \approx 10^{-7}$ calculated for a bursty channel model. The

non-bursty channel, discussed in the previous Chapter, achieves an output symbol error rate of $P_{symb,non-bursty} \approx 10^{-11}$. This means correction rates can vary up to 4 orders of magnitude depending on the characteristics of the channel errors.

The decoder choice influences the results by about one order of magnitude in favour of the more modern one, decoder II. The same result was found for a non-bursty channel. In the following decoder II will be used only.

5.4.4 Error Correction Capacity on a Bursty Channel and Memoryless Channel

To summarise, symbol error-correction rate of both non-bursty and bursty channel, as introduced in paragraph 5.4.2, are presented in Figure 5.8. The graph shows interpolation rate and symbol error rate of both channel models corrected by decoder II. The author's computer simulation proves a weakened correction ability of a bursty channel compared to a memoryless channel, as expected.

The diagram suggests that towards lower input error rates the error correction is worsened compared to non-bursty errors. Blocks of erroneous symbols cause more problems for the error correction system than randomly distributed errors. Input error rates above $r = 0.1$ lead to the same correction rates. Above this level erroneous symbols form enough accumulations to cause the error correction to fail like on a bursty channel.

5.4.5 Discussion

It can be seen that the character of error distribution - either using a memoryless channel model or a bursty channel model - influences the capacity of the error correction

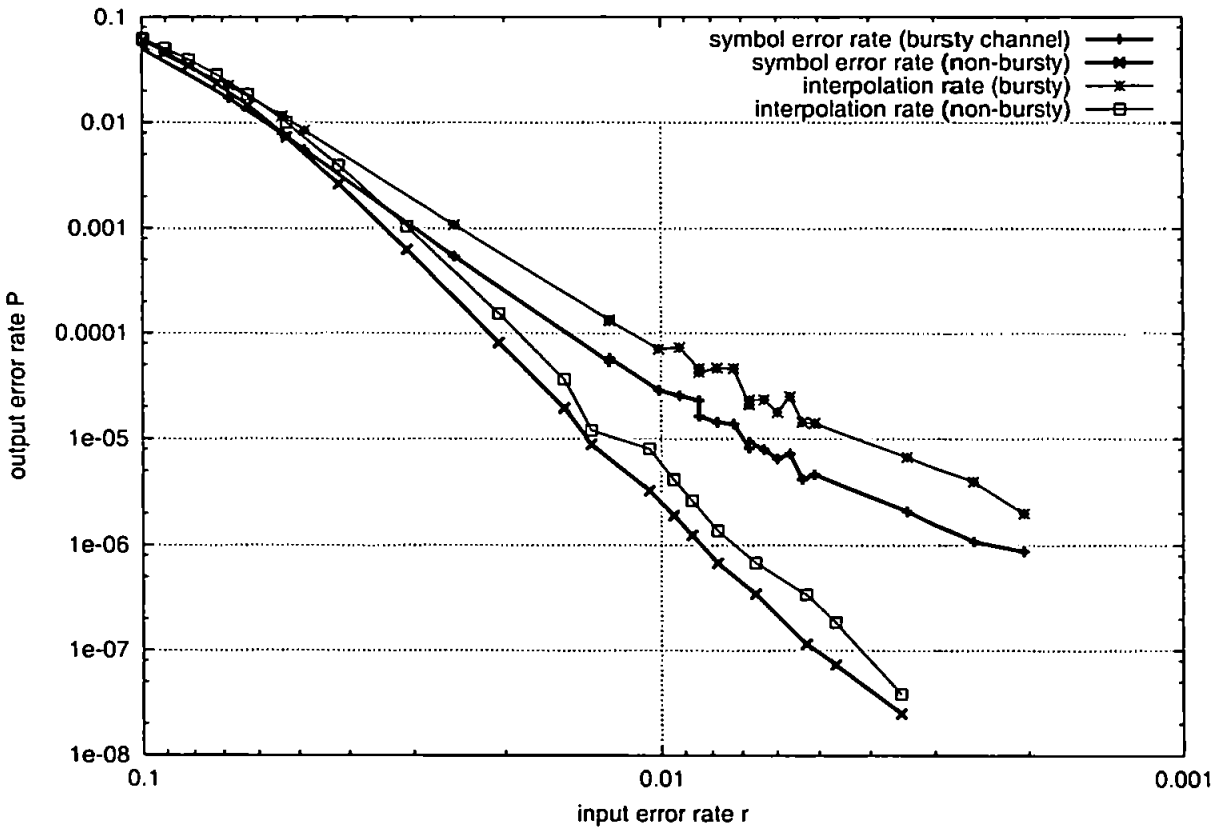


Figure 5.8: Decoder II applied to a non-bursty channel model and one with the distribution presented in paragraph 5.4.2. Towards lower input error rates the difference is evident and can be extrapolated to the deviation of several magnitudes.

decoder massively. Two decoding strategies were checked against a memoryless channel and a bursty channel. Both of them reveal differences in error correction capacity.

In this Chapter an analysis has been done for bursty channels. The result shows that random, statistically independent erroneous symbols in the input stream cause much less trouble to the error correction system than sequences of errors.

The simulations of the bursty channel were based on a special error distribution. Applying decoder II to these real-world distribution resulted in different types of error rates. Investigations in this area have not yet been published. The motivation for emulating experimentally measured error distributions as opposed to either random errors or simple random burst errors [33] was to evaluate watermarking schemes as closely as possible to a real-world scenario. The idea of simulating error distributions in this way is new though and has brought quantitative results. The next Section focuses on testing of certain watermarking schemes, considering a normal user-handled disc.

5.5 Performance of Watermark Sequences

5.5.1 Introduction

Any new information added to the data stream must be placed in known positions in order to allow this data to be recovered. This recovery should take place before the error correction is invoked. Ideally the error correction system fully corrects the original information, though weakening further error correction capacity for random, naturally occurring errors.

The allocation of certain symbols in a frame for watermarking purposes has effects on the error-correction rate. Choosing different patterns and places for the introduced symbols might result in a better performance of the error correction system.

Introduced error patterns are characterised by their generation function and the *introduced symbol error rate* (r_{intro}) denoting the error rate of the intentionally introduced error symbols. The capability of the error correction can be expressed by either the distinct terms: $P_{00}, P_{01}, P_{10}, P_{11}$ or additionally one can consult the rates discussed in paragraph 5.1.2 The one to choose depends on the purpose and interest of the investigation. In this work two parameters deserve a closer look: The symbol error rate and the interpolation rate.

The symbol error rate P_{sym} equals the sum of $P_{10} + P_{11}$ and counts the number of uncorrected symbols plus the number of miscorrected symbols. The higher this value, the less a certain pattern is correctable.

The interpolation rate P_{ip} expresses how often the interpolation of audio samples is invoked. In the case of a CD-audio player it means that the quality of the reproduced sound is deteriorated if this value is high.

For other parameters, such as P_{10}, not enough symbol errors (less than 40) are generated to get reasonable reproducible results in 250000 iterations. To compensate for this, more iterations are necessary, which would multiply the computing time.

It is also of interest which effect the used probability density function of the background noise has on the overall error correction rate. This will be discussed in the next Section 5.5.2. Since the aim is to find a pattern the least prone to errors and to predict quantitatively the influence, the most realistic case is assumed. Therefore the error distribution function of Figure 5.5 is employed modelling a user-handled compact disc.

The rate of introduced additional error symbols typically varies from $r_{intro} = 0.005$ to $r_{intro} = 0.3$, which is less than one wrong symbol per frame to a third of a frame rendered bad. Apart from generating intentional errors, a second process is responsible for generating the erroneous symbols simulating the background noise. These two processes work independently of each other.

A number of different error pattern generators has been tested against a bursty background noise distribution, presented in Section 5.4.2 (Figure 5.5c). In the following each of them is presented with its results in terms of resistance against error correction. For each different combination of parameters, 250000 frames (iterations) have been simulated. This establishes a compromise between computing time and a reasonable accuracy.

The x-axes of the diagrams (introduced error rate r) denotes the input symbol error rate for introduced erroneous symbols.

The y-axes (relative error frequency f) denotes the ratio between overall output error rate for introduced error patterns plus background noise and output error rate for background noise only. The overall output error rates are thus expressed as a

introduced error rate r_{intro}	0.005	0.02
bursty noise: output error rate $P_{sym,bursts}$	$5.45 \cdot 10^{-5}$	$1.75 \cdot 10^{-3}$
random noise: output error rate $P_{sym,random}$	$1.15 \cdot 10^{-7}$	$8.15 \cdot 10^{-5}$
ratio $x = \frac{P_{sym,bursts}}{P_{sym,random}}$	473.9	21.5

Table 5.2: Output symbol error rates $P_{sym,random/bursts}$ of non-bursty, random background errors and bursty background errors, combined with systematically induced symbol errors of two different error rates r_{intro}. The ratio shows that random and bursty noise have different correction rates P_{sym}, as expected, but also that the ratio of correctibility is depending on the rate of induced errors.

multiple of the error rate of background noise only.

5.5.2 Motivation for Considering Background Noise

Bursty noise, as it is present in user-handled discs, has a lower error correction rate than random noise. Table 5.2 lists values for the output error rates P_{sym} of an error distribution composed of background noise and introduced symbol errors. Due to the better correctibility of random noise, the ratio is $x \neq 1$. In addition, x is depending on the introduced error rate r_{intro}. This is explained by the fact that with higher introduced error rates this part of the overall input error distribution overweighs the background error distribution.

It is believed that the ability to correct introduced errors on a noisy channel depends on the type of background noise. Thinking of an introduced double error each frame, it is possible that the underlying noise probability function gives rise to different correction rates for that introduced error.

In order to illustrate this behaviour, combinations of bursty background noise, non-bursty background noise and systematically induced two-symbol error patterns have been simulated. Table 5.3 lists the number of symbol errors after decoding 250000

frames.

characteristic errors present	bursty background	plus induced two-symbol error
resulting symbol errors n	10824	79816
characteristic errors present	random background	plus induced two-symbol error
resulting symbol errors n	2400	68777

Table 5.3: After decoding 250000 frames the number of symbol errors n varies depending on the form of noise present. The induced error rate is $r_{intro} = 0.0625$, which is a two-symbol error per frame.

Based on Table 5.3 the ratio between correction rate of background noise only and background noise plus additional, intended symbol errors, for the two types of noise, has been compared in Table 5.4.

$\frac{n_{bursty+induced}}{n_{bursty}}$	7.37
$\frac{n_{random+induced}}{n_{random}}$	28.66

Table 5.4: Considering the values derived from Table 5.3 it becomes evident that introducing a two-symbol error per frame does not have the same effects on the output correction rate on bursty and non-bursty background noise.

It becomes evident that introducing two-symbol errors on a bursty background has less effect on the output error rate than introducing errors on a random background noise. Actually the output error rate increases thereby by a factor of 28, whereas the output error rate on a bursty channel increases only by factor 7. This is a four times difference of the increase of error rates when inducing additional errors due to the type of background errors.

It is therefore important to consider background noise in general, and in particular to precisely model the error distribution. Introduced errors form a particular error distribution, too, even though the distribution is discrete. The superposition of a specific background error distribution and introduced error distribution is therefore crucial for the error correction rates. Effects of introducing erroneous symbols in terms

of the overall output error correction rate thus depend on the type of background noise present. These variations of output error rates in turn are specific for different patterns of intentional errors.

Therefore the calculations for the next chapters are based on the presence of background noise, as specified in Figure 5.5.

5.5.3 Using Intentional Errors as Watermarks

Equidistant Symbols Pattern (EQU)

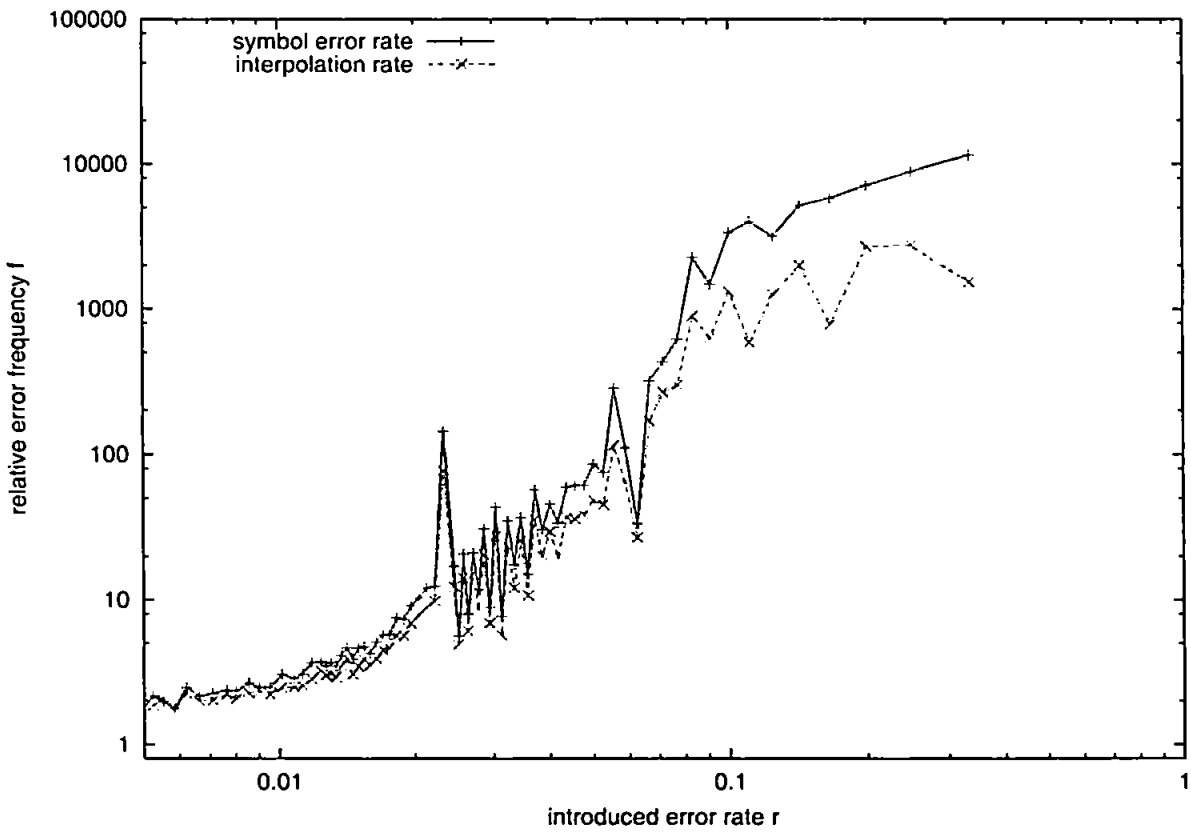


Figure 5.9: Correctibility of equidistant error symbols (EQU) together with background noise. The relative error frequency f is based on the correction rates for background noise only.

Every n symbols one wrong symbol is entered. The minimum distance between two

consecutive n is 2, the maximum is $n = 200$. The minimum input symbol error rate is $r_{intro} = 0.005$ for $n = 200$, the maximum is $r_{intro} = 0.3$.

This scheme is one of the most simple to apply. The correctibility in terms of single errors and sequence interpolation rate is shown in Figure 5.9.

Some error rates show high peaks in the output error rate. In these cases patterns are created for which the error correction process cannot cope as intended. The opposite can happen, too. The distance $n = 16$ and $n = 32$ marks a relatively good correctibility probably due to $n = 15$ separating each frame into two equal parts. Distances of $n = 30, 31, 35$ ($r_{intro} = 0.029 \ldots 0.033$) result in low symbol error rates (single peaks), whereas $n = 42$ ($r_{intro} = 0.024$) results in a ten times higher overall error rate (one peak).

Sequences of Wrong Symbols Pattern (ESE)

One sequence of wrong symbols is introduced at the beginning of each frame. The length of the erroneous sequence varies from 1 to 16 symbols in each frame giving an error rate of $r = 0.031 \ldots 0.5$. Results can be viewed in Figure 5.10.

Sequences of Shifted Wrong Symbols Pattern (ESS)

A variation of the above mentioned method is shifting the error sequence by one symbol each new frame. This distributes the error sequences more uniformly in the data stream (see Figure 5.11). Minimum is again one symbol per frame, maximum is 16 introduced symbols per frame. Figure 5.12 shows the result. The relative error frequency is above $r = 10$, conclusively the error correction ability for this pattern is worse than without shifting.

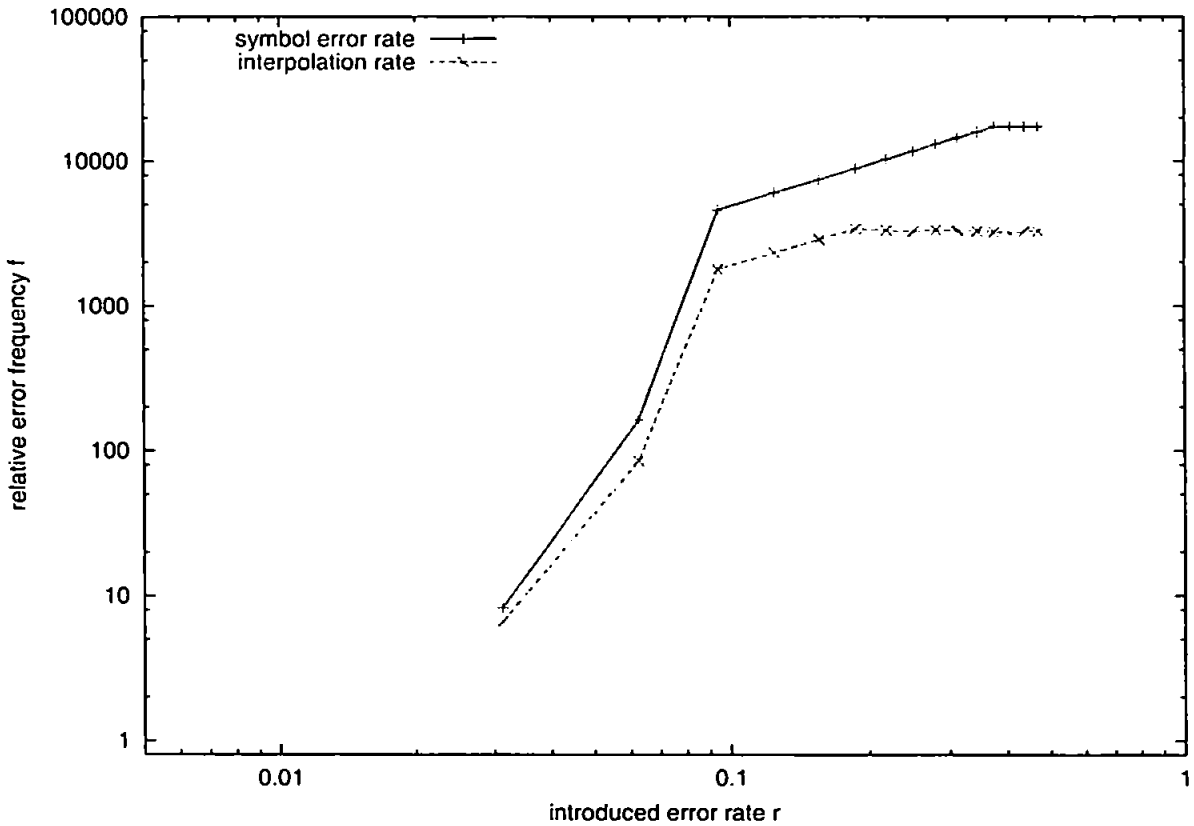


Figure 5.10: Correctibility of a sequence of errors with varying length at the beginning of each frame (ESE) plus background noise based on error rates of background noise only.

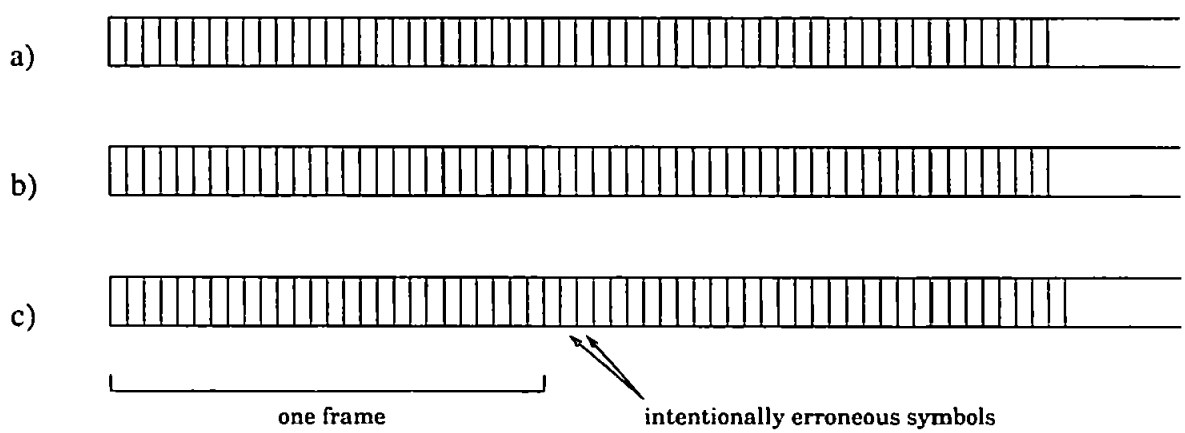


Figure 5.11: Schematic illustration of a shifted sequenced error pattern (ESS) (refer to Figure 5.12).

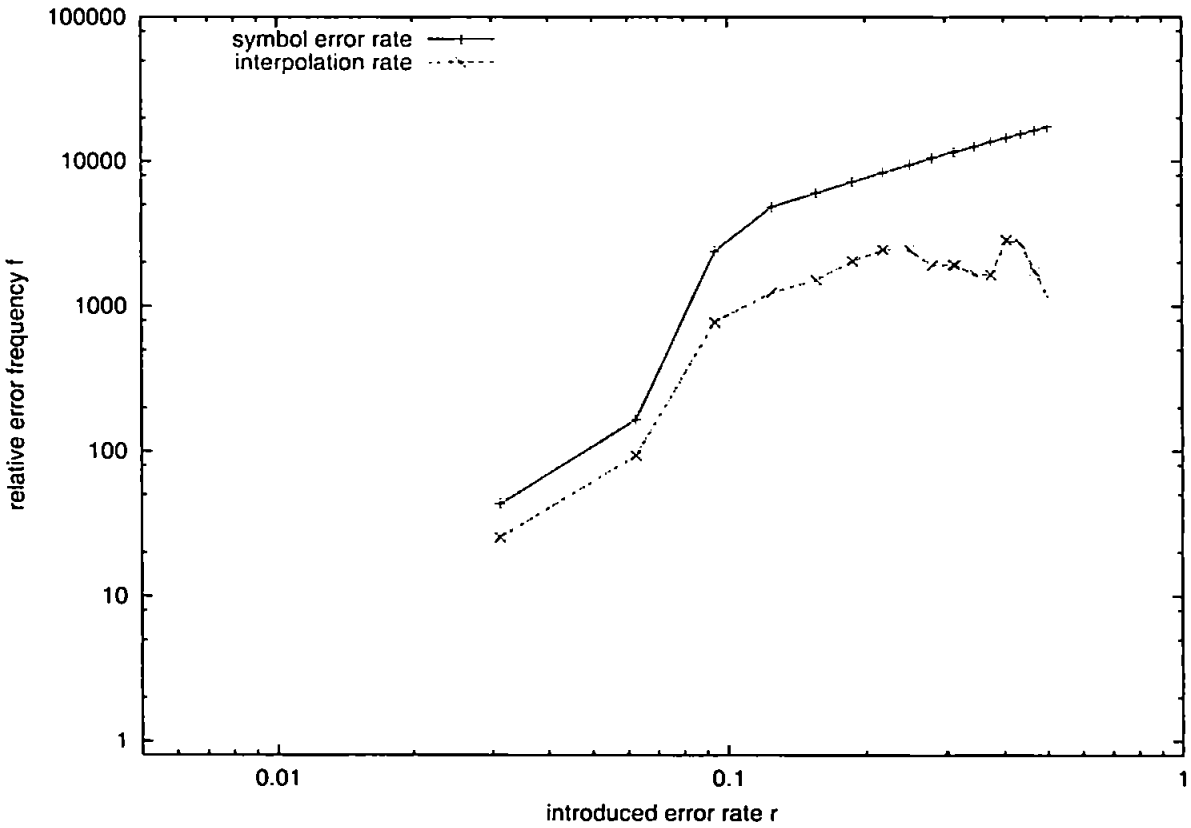


Figure 5.12: Correctibility of a pattern with shifted sequences of erroneous symbols each frame (ESS) plus background noise. The relative error frequency is $f > 10$ pointing to a lower correctibility than without shifting (compare to Figure 5.10).

Errors Grouped in Small Units (EGR)

In order to achieve a more uniform distribution of introduced errors over a frame, groups of two erroneous symbols are formed and allocated in equidistant places over a frame. The frame layout can be seen in Figure 5.13. This pattern performs well at low error rates, since for $n = 1, 2$ symbols it is the same as discussed two paragraphs previously. The correctibility of said pattern decreases rapidly at error rates of above $r = 0.07$.

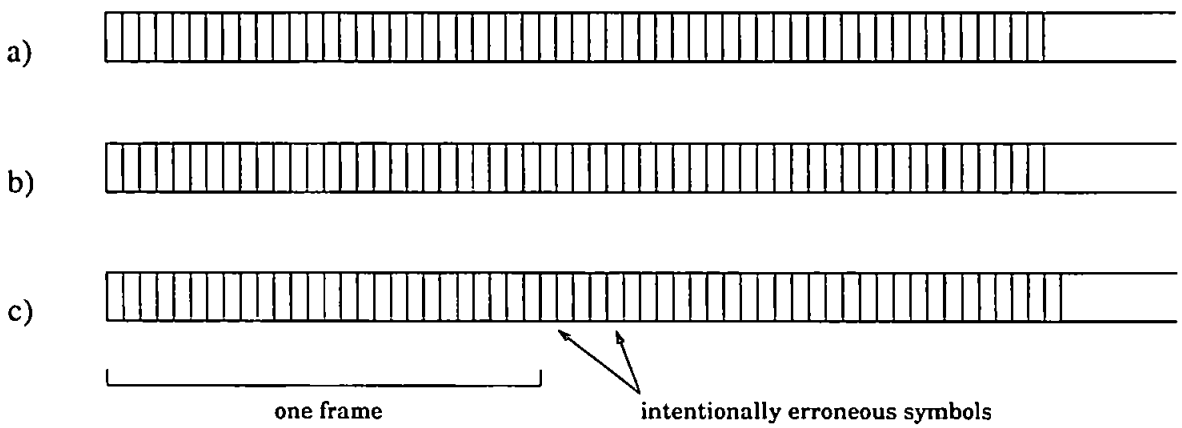


Figure 5.13: Schematic view of error groups distributed over a frame (EGR).
a) 2 symbols per frame. b) 5 symbols per frame. c) 12 symbols per frame.

Equidistant Erroneous Frames (EQF1/2/3)

Another way to allocate storage is in using up all symbols of one frame and interleaving these marked frames by a number of good frames. A schematic diagram of the layout is shown in Figure 5.16. It is expected that the error-correction ability is less than in all other cases, because long sequences of errors impose high requirements on the interleaving and scrambling system. Figure 5.15 shows the resulting error probabilities. The curve shows big variations around a relative error frequency of $f = 100$ for $r > 0.03$. The lowest error rate introduced by a gap of 200 good frames between one bad frame

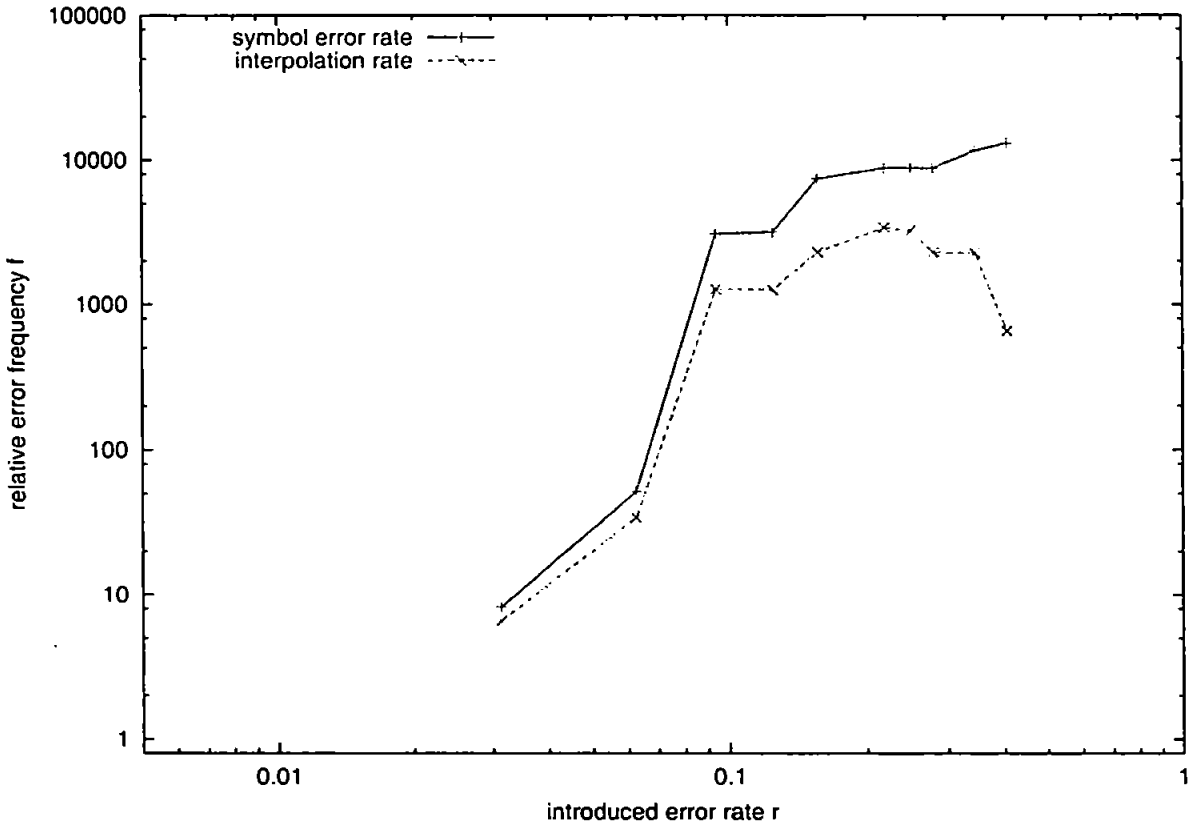


Figure 5.14: Correctibility of groups of error symbols (EGR) plus background noise. For low error rates $r < 0.07$ the pattern is identical to the one used in Figure 5.10.

is $r = 0.005$.

It becomes evident that the scrambling function cannot spread the erroneous symbols well enough to achieve the low corrected error rates obtained with previous patterns.

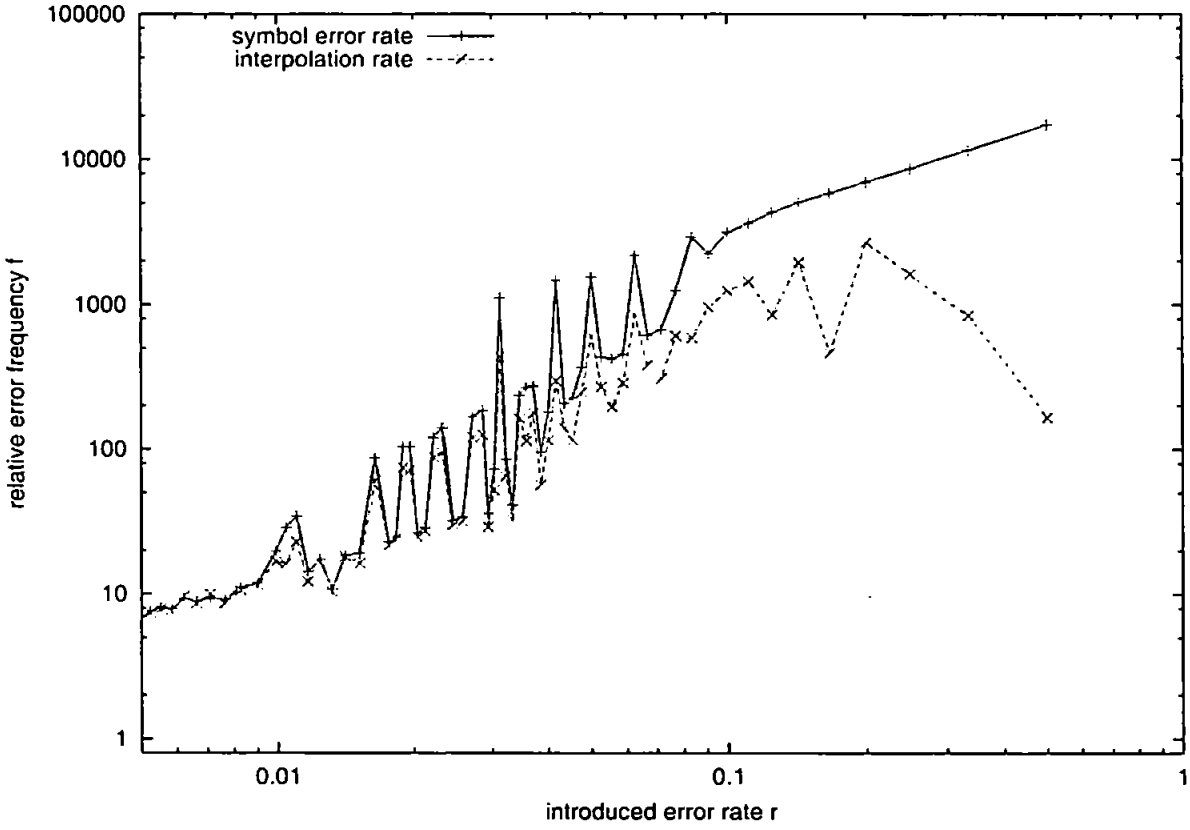


Figure 5.15: Correctibility of a pattern with alternating error-free and erroneous frames (EQF1) plus background noise.

The discussed pattern can be further thinned by allocating every second symbol in one frame to an error. A variable number of intermediate frames are error-free. The layout is shown in Figure 5.18. The obtained error rates match the previous case and are not favourable in terms of correctibility.

By allocating every third symbol to an error, the layout in Figure 5.20 is obtained. Figure 5.19 shows the resulting correction capacity. Like with the other frame patterns

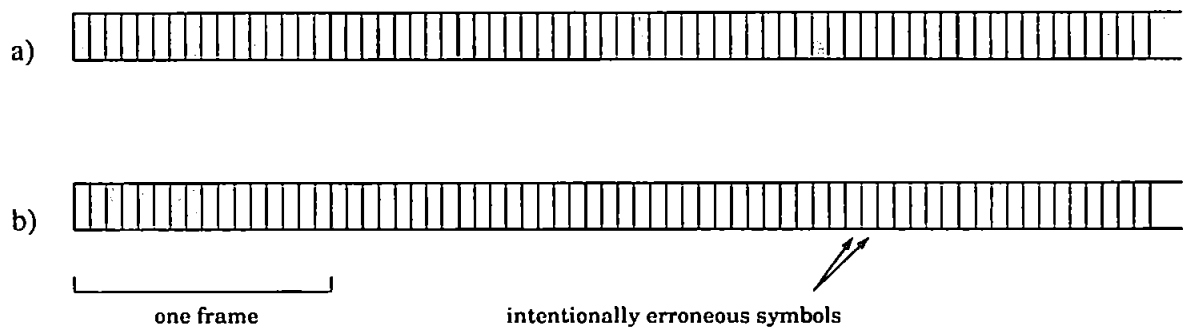


Figure 5.16: Equidistant erroneous frames, the intermediate gap length is variable. a) one frame distance between one erroneous frame. b) two frames distance between one erroneous frame.

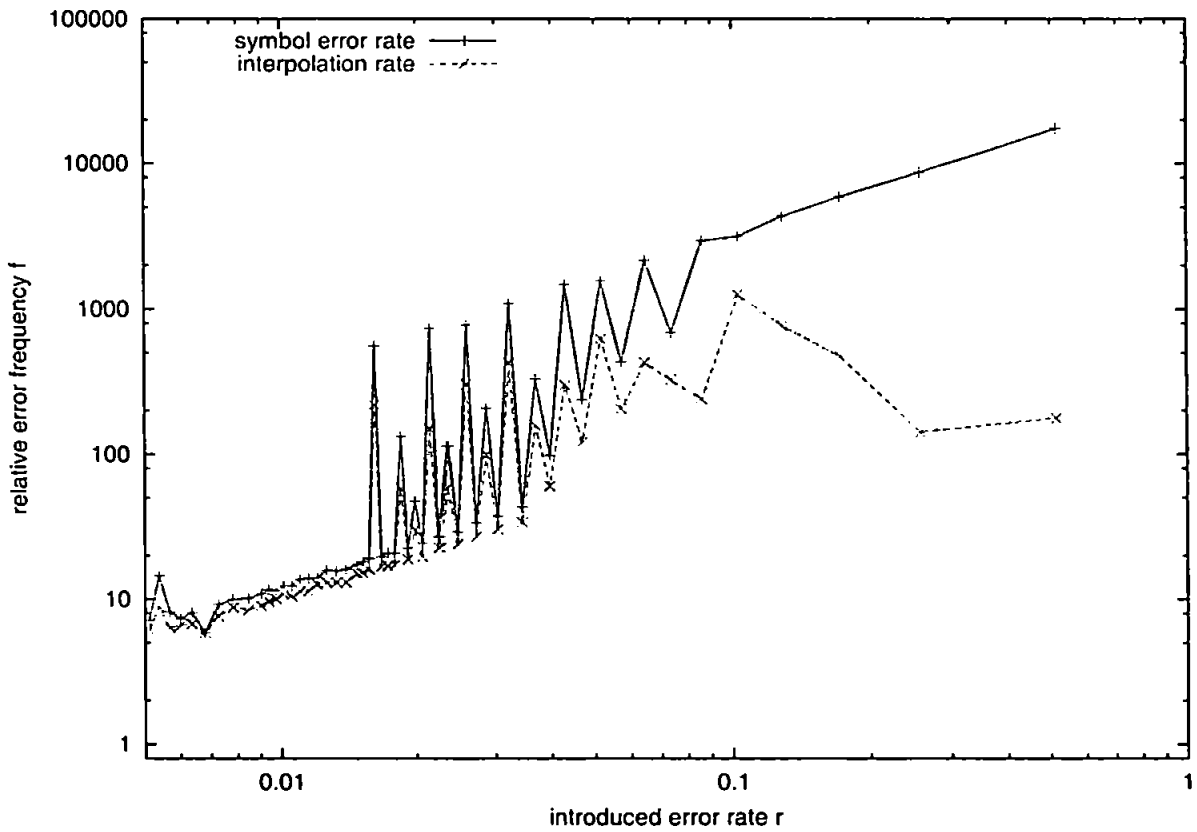


Figure 5.17: Correctibility of a pattern of alternating error-free and erroneous frames plus background noise. The erroneous frames are filled with alternating error and non-error symbols (EQF2).

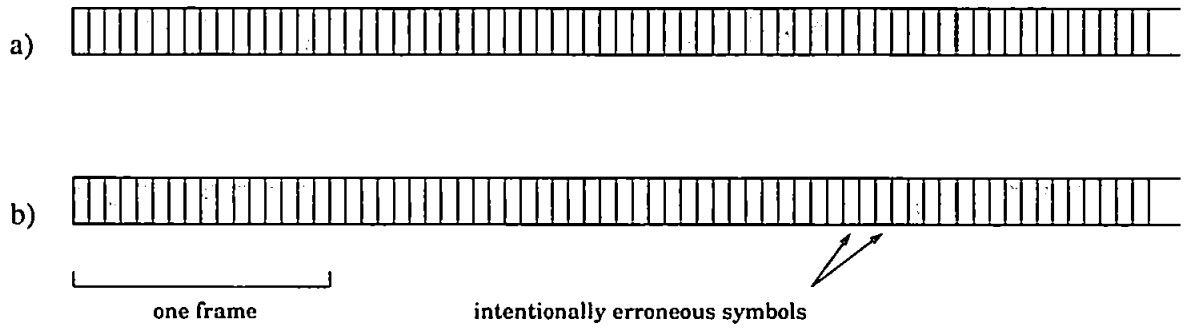


Figure 5.18: Equidistant erroneous frames, the intermediate gap length is variable. The erroneous frames are filled with alternating error and non-error symbols.
a) one error-free frame distance between one erroneous frame. b) two frames distance between one erroneous frame.

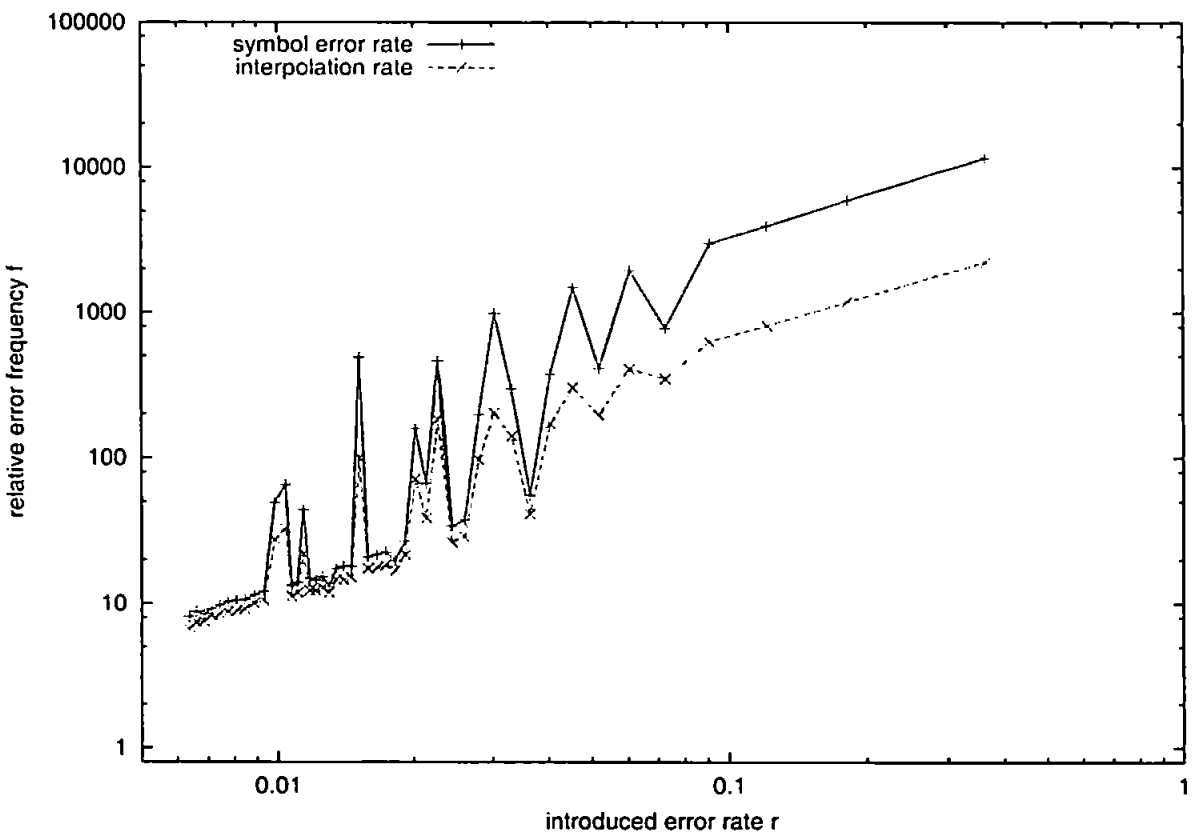


Figure 5.19: Correctibility of a pattern of alternating error-free and erroneous frames plus background noise. The erroneous frames are filled alternatingly with one error and two non-error symbols (EQF3).

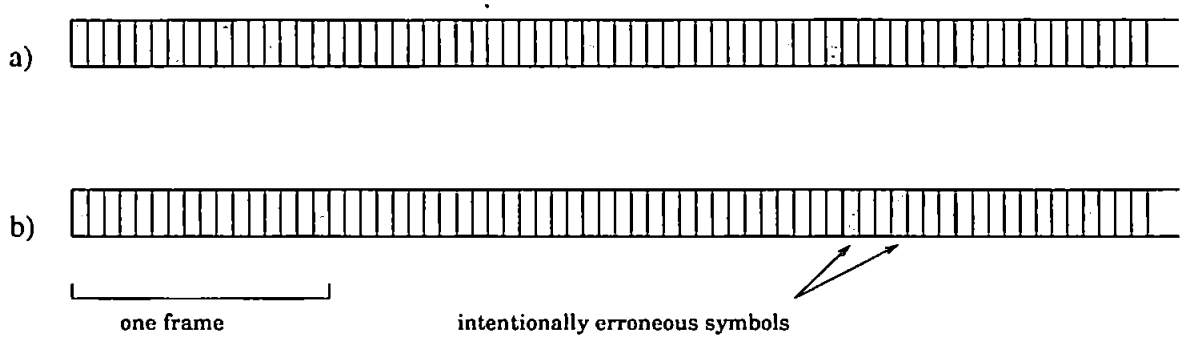


Figure 5.20: Equidistant erroneous frames, the intermediate gap length is variable. The erroneous frames are filled with alternating one error and two non-error symbols. a) one error-free frame distance between one erroneous frame. b) two frames distance between one erroneous frame.

the output error rate can vary by about one order of magnitude in consecutively steps, though the overall performance is bad.

Randomised Intervals of Erroneous Symbols (RES)

Motivated by the fact that non-bursty, random background noise on a channel is more correctable than bursty background noise, a pattern of randomly distributed wrong symbols has been investigated. A pseudo-random number generator therefore produces a sequence of random numbers which represents the distances between the wrong symbols.

The random number sequence can be reproduced by feeding the random number generator with the same seed. The generation routine is taken from [43] and outlined in Figure 5.5. The algorithm is easy to use and very fast, producing sufficiently good random numbers.

Figure 5.21 shows the result. The curve shows a typical behaviour due to the randomness of the symbol allocation. This type performs worse than expected. Comparing Figure 5.9 and results from this paragraph, a conformance is obvious. Equidistant er-

```
float rand;
unsigned long idum, itemp;
static unsigned long jflone = 0x3f800000;
static unsigned long jflmsk = 0x007fffff;
idum = 1664525L * idum + 1013904223L;
itemp = jflone | (jflmsk & idum);
rand  = (*(float *)&itemp)-1.0;
return rand;
```

Table 5.5: Fast random number generator for distributing erroneous symbols.

ror symbols are a special case of a random distribution. Therefore the relative error frequency match each other, even if there are peaks in the previous one resulting from certain sequences which the interleaver accumulates in a way that is hard to correct.

Comparison and Discussion

Figure 5.22 and 5.23 summarises all eight curves. There are three cases to distinguish. Each of them shows its own characteristic slope and the extrapolation of each of them approaches the same limit. The first case includes patterns with whole erroneous frames (EQF1/2/3). The second group encompasses equidistant (EQU) and random symbol patterns (RES) and the third group patterns with symbol errors in the same place relative to the frame beginning (ESE/EGR). Shifted patterns (ESS) are a special case of the third which perform a lot worse.

It is found that using the same locations within all frames for inducing errors is the best to correct. The relative error frequency $f = 0.03$ performs comparably well when using the same positions within a frame. Other patterns perform about three orders of magnitude worse at the same introduced error rate r, which is remarkable.

Despite this it must be noted that when marking one symbol in a frame erroneous the output error rate depends on the position within the frame. The error rates shown

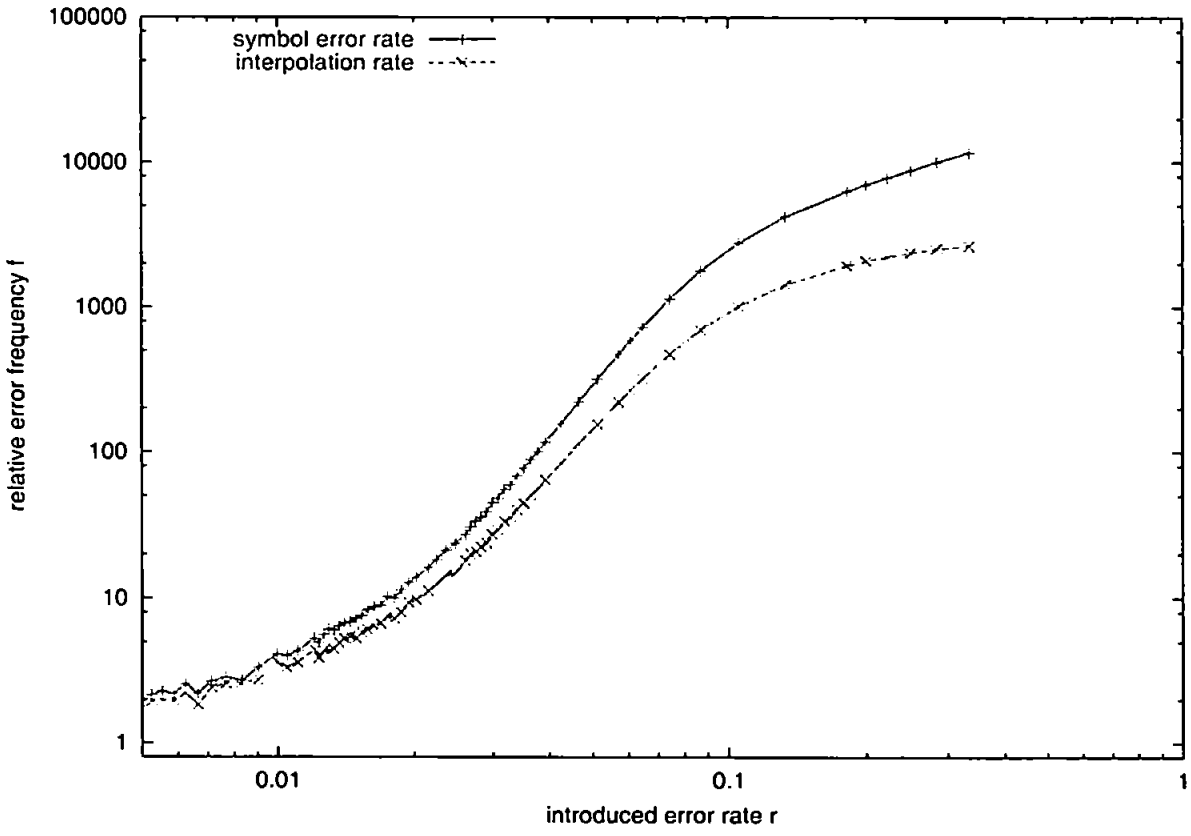


Figure 5.21: Correctibility of randomly distributed one-symbol errors (RES) approximating the lower limit of $f = 1$ very slowly.

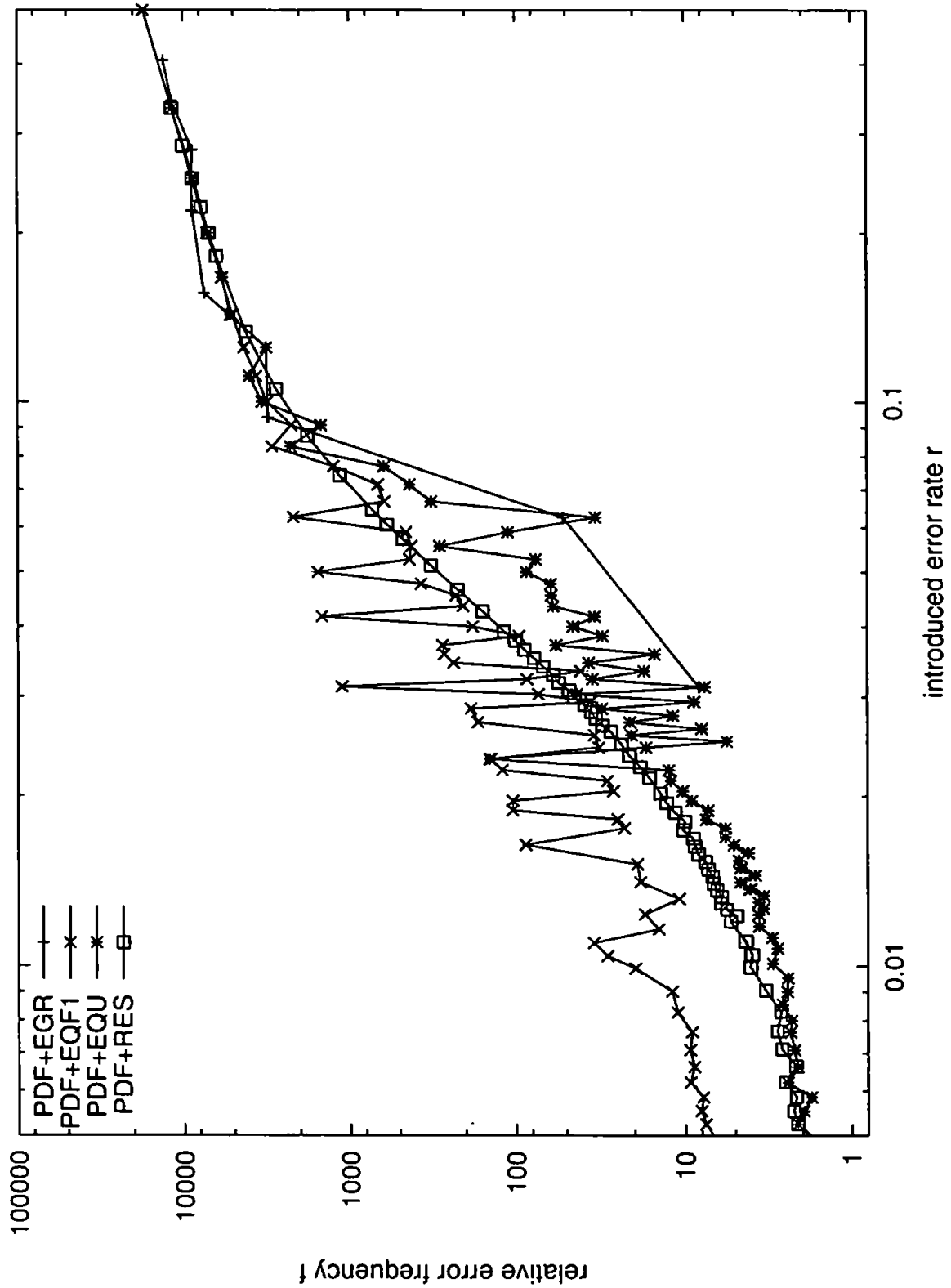


Figure 5.22: Summarised overview of four error patterns. The notation "PDF" means bursty background noise is added. The other notations refer to the type of pattern used. Three different groups are recognisable with different types of approximation. The next four patterns are shown in the next diagram, Figure 5.23.

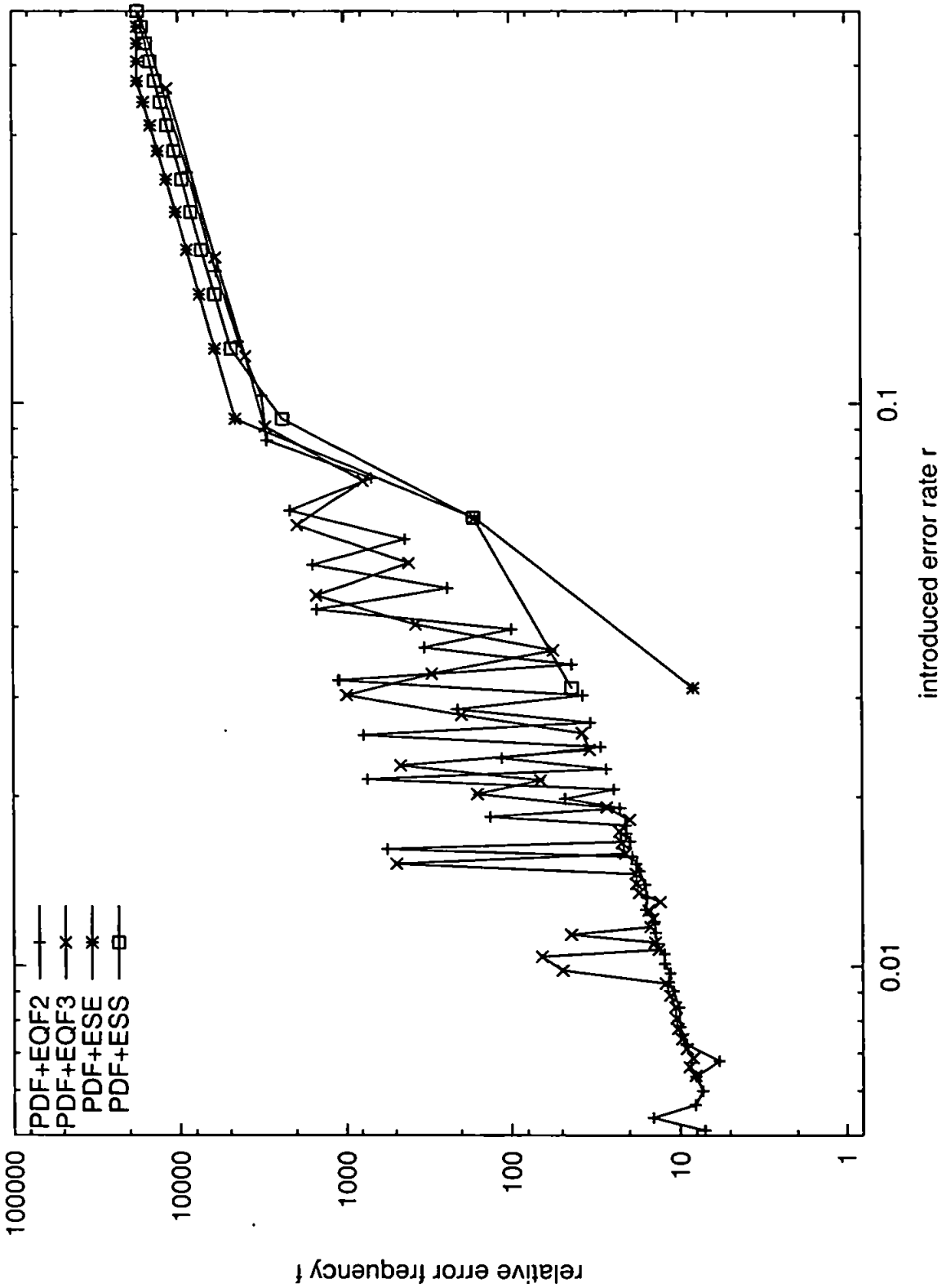


Figure 5.23: Set of the next four error patterns (continued from Figure 5.22).

are valid only for introducing a symbol error at the beginning of each frame. Further simulations prove that according to the position within a frame the error rates vary slightly.

It was assumed that the background error rate is $r_{background} = 0.01$, which is about a hundred times higher than error rates found on real discs [47]. This assumption was necessary to run the simulation in a reasonable time. It is believed that the introduced error rates r_{intro} must be changed accordingly to lower values in order to get the same relative error frequency f. All patterns approximate $f = 1$; the better the correctibility, the faster is the approximation.

The minimum of the ratio relative error frequency f to introduced error rate r $\frac{f}{r}$ identifies the point where it is most economical to introduce an error pattern. Differences of more than three orders of magnitude between certain patterns on their lowest ratio points out again the fact that an optimised way for introducing watermarking data exists. Figure 5.24 and 5.25 presents the results.

To summarise it can be postulated that putting data as part of watermarking information in the same location in each frame is the most promising way. Some locations though are better suited compared to others. By interleaving this pattern with good frames, lower error rates can be achieved, still performing better than other types of patterns. Accumulating erroneous symbols in more than two successive symbols lets the correction rate drop.

5.5.4 Choosing non EFM-Words as Error Symbols

The basic principle of this watermarking scheme using error correction is data hiding in certain locations rendering the specific symbol erroneous.

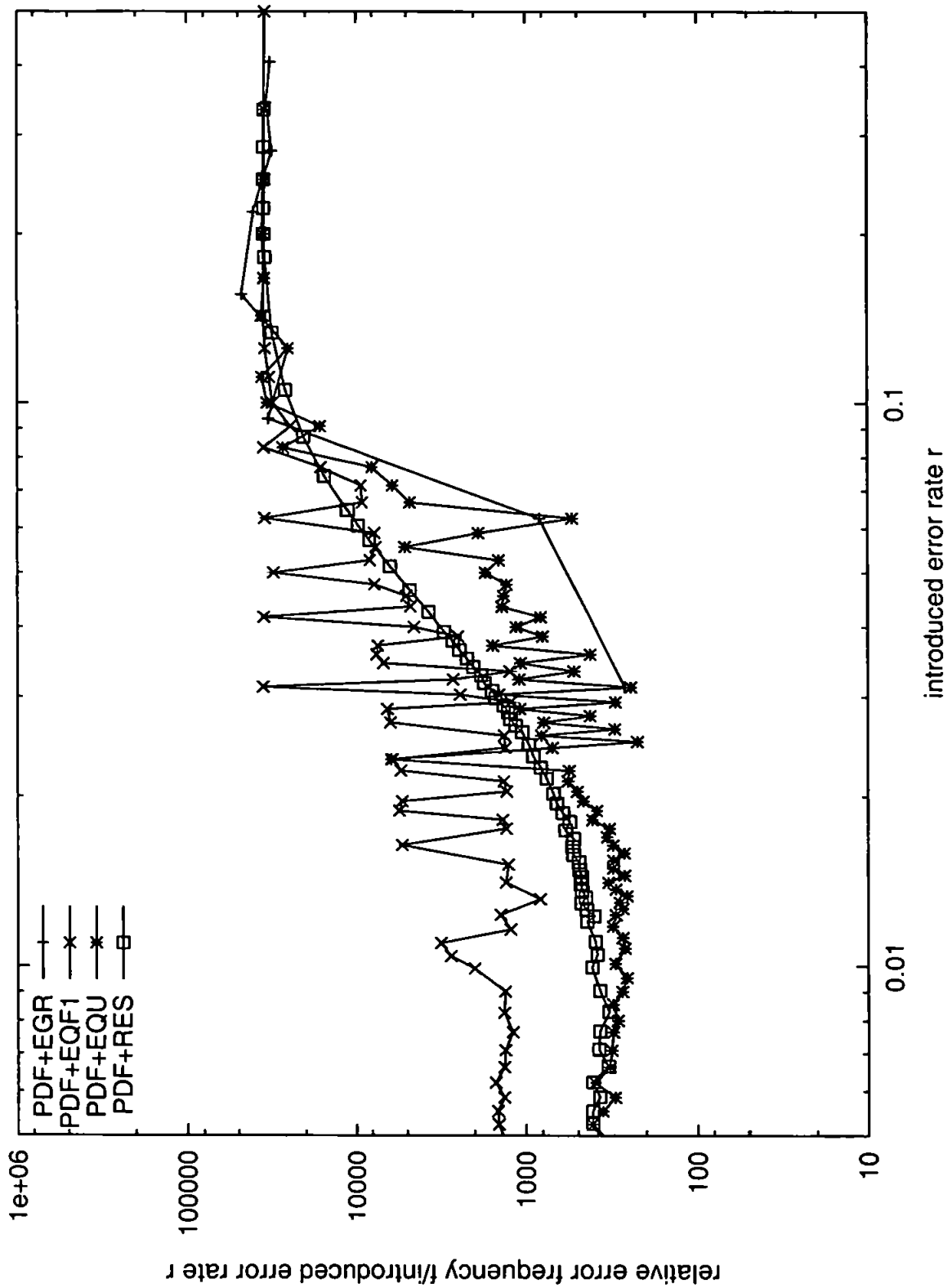


Figure 5.24: The relative error frequency f is standardised to the introduced symbol error rate r. The minimum signifies a good ratio of correctibility to information content. Four error patterns are shown here, the next four in the next diagram, Figure 5.25.

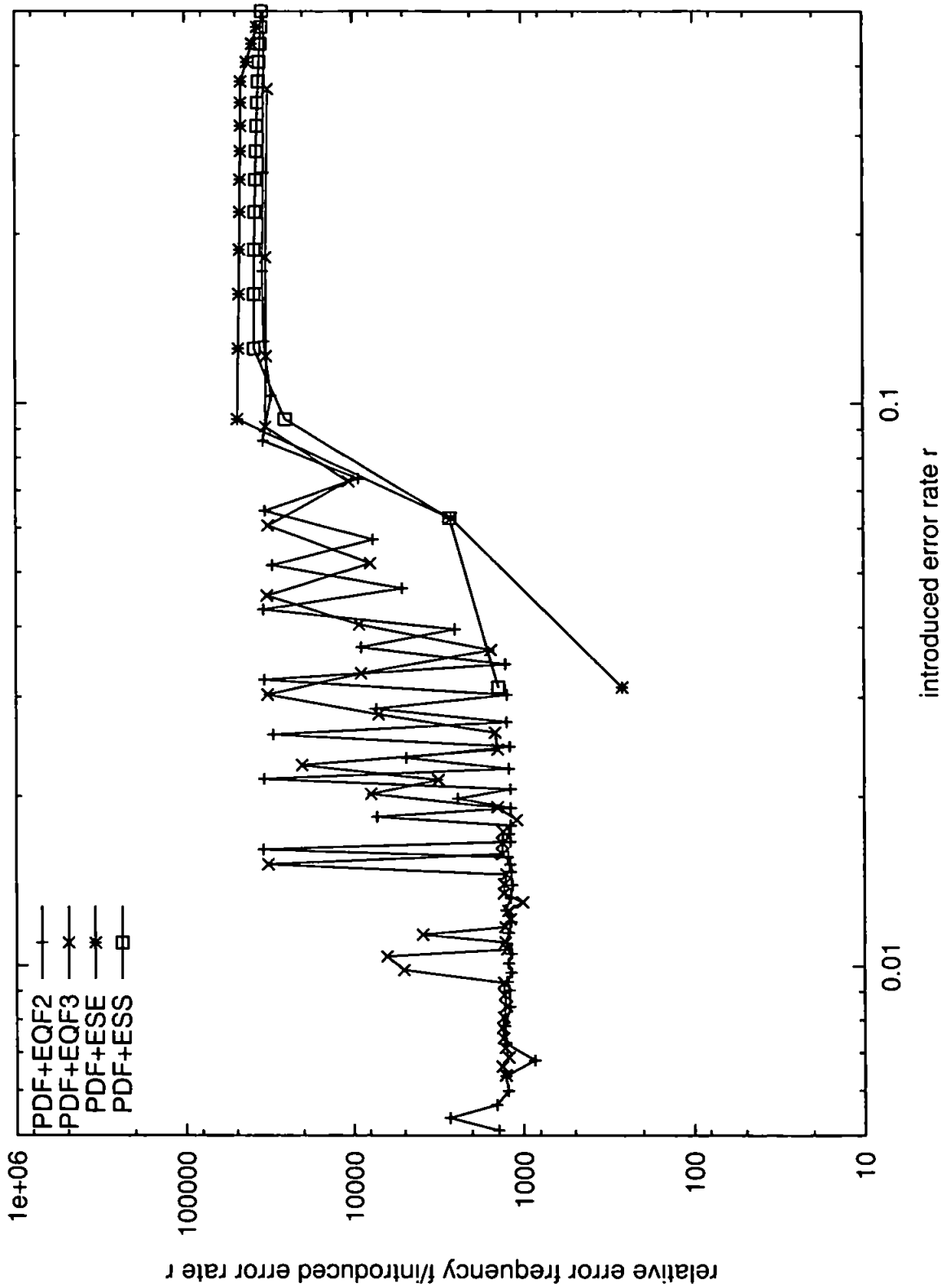


Figure 5.25: Set of the next four patterns, relative error frequency f standardised to the introduced error rate r (continued from Figure 5.24).

It is questionable whether the error values should be restricted to valid EFM-symbols only or whether they can include non-EFM words as well. Allowing non-EFM symbols certainly increase the probability of failure to read a track successfully, especially in the case of a high watermark information content, due to the disruption of the run-length-limited code. But based on the fact that the goal is to achieve only a moderate watermark information content, it is feasible to allow for non-EFM symbols as errors. Assuming the watermark content is detected before the EFM-conversion to 8bit symbols, the information content of one erroneous symbol is 14bits (without 3 merging bits).

A variation in correction rates is possible by carefully choosing the error symbols. If error symbols are chosen not to be EFM-codewords, the EFM-converter reports a possible error on this location to the error correction system by flagging this symbol as an erasure. This increases the capability to correct this one symbol.

Choosing the error values out of all possible variations reporting valid EFM codewords becomes unlikely. Since there are 256 valid EFM-codewords out of 16384 (2^{14}) the difference is about 1.5% and can be neglected. The author's opinion is, therefore, that it is not of importance to restrict error values to non-EFM symbols.

5.5.5 Changes to the Overall Error Rate when Introducing Watermarks

Considering an underlying noise level, the question is how much error correction capacity a typical watermark pattern uses up. Therefore the author computed the output error rates of bursty background noise only over a range of an input error rate of $r_{PDF} = 0.002 \ldots 0.02$ (similar to Figure 5.8 with a higher number of frames decoded

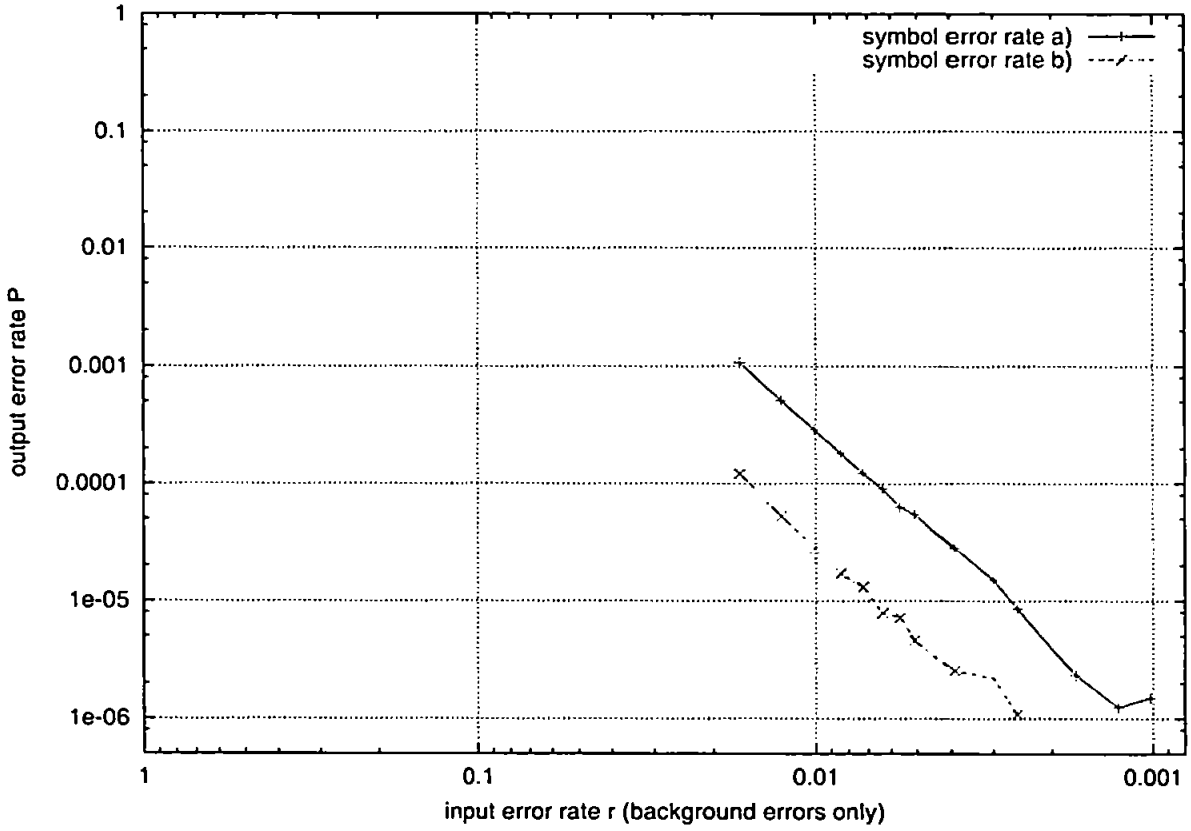


Figure 5.26: Output error rates of background noise with varying input error rates r if a) a regular watermark pattern is present and b) only background noise is to correct.

(one million frames)). Secondly output error rates have been calculated but with an additional one symbol error each frame. This obviously lowers the output error correction rate. Figure 5.26 presents the result.

The output error rates ranging from $r = 0.002 \ldots 0.02$ continuously are about one order of magnitude lower for background noise only. It is important to bring to mind that this holds true only for the characteristic burst distribution used to model the background noise. According to Section 5.5.2 a different noise distribution would give a different result.

The second aspect of this outcome is how much the background noise level must be reduced in presence of a special watermark pattern in order to obtain the same output error correction rate. It can be seen from the diagram that lowering the noise level by about 60%, the same output error rate is achieved.

5.5.6 Conclusions

This Chapter focussed on adding data deliberately in certain locations in the encoded data stream in order to extract them at a later point of time before the error correction takes place. The error-correction algorithm will then recover the original data. There are a number of conclusions that can be drawn from this research.

First it is to be stated that the generator function for the locations matters to a great extent. Different ways of allocating places for additional data vary in their performance in terms of the output error rate over a range of almost three orders of magnitude. This is unexpected, because it leads to the conclusion that terms like "block error rate" or simply "error rate" on its own are not meaningful. Instead the systematic distribution and characteristics of the errors have to be considered.

Indeed these error patterns can be generated naturally by a faulty compact disc drive. A possible scenario would be that ageing hardware components generate systematic errors. One option is that the synchronisation of frames are lost sometimes resulting in whole frames not read correctly. The other option causes some symbol data to be wrongly restored from the signal by the electronic circuitry, which appears to be a random process. Clearly it can be proved that the latter would cause less trouble for the error correction system to cope with.

Every compact disc drive makes errors in recovering the original data from the laser signal. The effects on the error correction performance depends, according to this work, on the type of these errors, whether they are randomly distributed or have a more systematic nature. Errors with a very systematic behaviour (i.e. always the first symbols in one frame are not read correctly) are therefore up to ten times more correctable than random errors.

Naturally occurring scratches and dust on the surface of a compact disc obey a certain density distribution. They characterise the disc in a certain way, so that the density distribution can mean a difference to the applied watermarking patterns.

Assuming a process is supposed to introduce mostly radial scratches (wiping the surface from the inside outwards), it becomes apparent from the results that these scratches are not troubling the error correction as much as tangential scratches. Tangential scratches are more likely to contain a higher quantity of frame errors, which have been simulated as whole frames being erroneous, for which the error correction system performs up to three orders of magnitude worse.

Information, encoded in surface markings, shall be unique to a compact disc, making digital fingerprinting possible. As shown in Chapters 4.4.2 and 4.4.3 it is feasible to use

two dimensional, macroscopic patterns on the disc's surface or on the disc's reflective layer. It is required that the markings have the property to cause the least possible overhead for the error correction system. Based on the simulation done in this Chapter, an evaluation of employed patterns is possible.

The software implementation of the compact disc encoding, done in this work, represent the basics for further research in this area. An assessment of different two dimensional structures is possible, considering the relative frequency of basic erroneous symbol patterns.

One of the original goals of this work focussed on creating a second embedded data channel hidden in the error correction information. This is a desired application, since secure physical media identifiers can be implemented in this way. Moreover this secure channel is inaccessible to the user and thus is perfectly suitable for applications in a DRM framework [11], like watermarking or implementation of tickets for copy-generation management. The possibilities are numerous and open new applications in the field of watermarking and copy protection measurements.

Chapter 6

Conclusions and Discussions

6.1 Conclusions

This work deals with compact disc technology. Due to its wide employment and upcoming requirements in the copy protection area, the aim was to investigate the introduction of techniques for ensuring the use of this technology according to copyright standards. The basic method hereby used was the modulation of the optical laser signal due to already existing or purposely introduced variations in the reflective layer, the surface of the compact disc or the transparent substrate.

The general idea was to increase storage capacity of a conventional compact disc by some amount, depending on the technique, in order to be able to use this additional space for security purposes. These can be digital watermarks, secure media identifiers or biometrical information. It is based on the fact that the compact disc with its built-in error-correction capability has capacity to correct also for intentional errors. About 23% of the compact disc's raw storage capacity is reserved for error-correction data (eight 8bit symbols over 588 channel bits). A part of this amount is then used to carry

additional user-data.

The author therefore conceived a technique by which modulations of the laser signal get mapped onto a rectangular grid, enabling further processing of the data like searching for existent watermarks, thus identifying a compact disc uniquely.

This technique is supported by specially written software which controls the acquiring of the data in an optimised way and evaluates the data with regard to steganographic information.

In a first trial an oscilloscope was employed tapping off the preamplified signal of the laser photo detectors. It was found that even if playing the compact disc normally without interruptions, the signal showed a lot more information in the form of signal modulations. These signal modulations seemed to be caused by variations in the reflection and absorbency of the compact disc. This was proved by purposely introducing distinct blemishes and covering the compact disc with small patches of opaque material, which was successfully matched on the oscilloscope to locations on the surface of the disc, while playing the compact disc without noticeable flaws.

In order to achieve higher resolution and to obtain an automated two-dimensional readout of the surface of a compact disc, an apparatus was built using a conventional compact disc reader device. The aim was to investigate compact discs for certain existing defects and to use this knowledge for an electronic characterisation of this disc.

During the testing phase it became apparent that each disc has its own surface map. This is caused mainly by the existence of stationary dirt and fingerprints on the surface of a user-handled disc. The readout system encompasses an electronic trigger with integrated amplifier to map positions on a disc and the ADC to convert the signal

to its digital form and store it on the computer's hard drive. The associated control software makes a fast and accurate read-out possible. This was achieved by using an accurate, high resolution software timer clock; the read-out task was running in a high priority state within the operating system. Specific sector or track read-out was established by using generic SCSI commands sent to the read-out device. Reports of possible read-out errors like track loss or CIRC errors had been reported back to the host computer.

The modulations of the reflected laser signal have been two-dimensionally imaged. It was found that apart from dirt on the surface a whole lot more data about the quality and consistency of the compact disc could be collected. The images revealed defects in the reflective layer during disc production as well as textures in the polysubstrate layer. The results were verified by using an optical and a scanning electron microscope. All three images could be matched against each other. The resolution of the author's image capture system was determined to be in the micrometer area ($15\mu m \times 1.6\mu m$). The number of greyscale levels depends on the ADC and the properties of the image capture system. Since there is a lot of high-frequency signal noise due to the laser reflection on the pits and lands, which cannot be matched accurately to any position on the disc, the number of grey levels amounts to about 100. In comparison, a computer monitor is able to display 100 grey levels. On the other hand, this method visualises accurately minuscule changes in the reflection coefficient of the reflective layer. Photographs taken with the optical microscope do not reveal details of a third dimension.

In [22] the authors introduce a similar technique to visualise surfaces with compact disc optics, although they do not rely on the laser tracking method applied in compact disc players. The resolution gained is similar to the author's system ($4\mu m \times 4\mu m$). The

captured pictures show similar properties. The need for a customised extra scanning mechanism makes this method less prone to track loss errors, but increases the costs for building.

Problems with the image capture include track loss and out of focus events. All naturally occurring blemishes like human fingerprints or manufacturing errors as long as they are reasonably distributed did not cause any problems. The image capture solution proved to be reliable, cost-effective and easy to implement in a common compact disc player due to few modifications necessary.

Commercial applications of a high-resolution, cost-effective apparatus to scan surfaces are possible. Since this technique relies on the existence of optical tracks to guide the read-out laser, only transparent samples can be visualised. This was demonstrated with fingerprints and surface scratches. Despite these shortcomings, the ease-of-use of the system makes it useful for certain niche applications.

Identifying compact discs by secure physical media identifiers is a lively topic discussed with respect to copy protection enforcement measurements [11, 4]. It is the aim of this work to suggest and evaluate watermarking and digital fingerprinting schemes for compact discs.

A provisional implementation of watermarking a compact disc has been presented. Two ways have been considered. Marking the surface of a compact disc with transparent blemishes modulates the surface reflectivity such that variations of the laser signal can be recognised and processed, making detection of watermarks possible. The other method involves puncturing the reflective layer, thus introducing accurate microscopic blemishes. An evaluation of the effects on the player's servo system showed that the blemishes should be less than $100\mu m$ in diameter in order not to cause damage to the

player's functioning. Defects down to $10\mu m$ have been resolved by the author's image capture system - the software was able to detect these blemishes. No work in this area has previously been published.

Detecting reflective-layer manufacturing defects, as stated earlier, provides a means to uniquely identify a certain compact disc. The detection of manufacturing blemishes with the author's system was backed up by optical microscope photographs of the same region. The research carried out in this work proves the feasibility of this technique in terms of hardware and software solutions for future copyright management systems [11].

Due to the fact that no crucial changes in the player's hardware took place (same read-out head, no additional amplifier for the reflected laser signal) it is the author's opinion that such a system can easily be implemented in today's compact disc and DVD players. Basically all the components are already present or can be added (AD converter, trigger).

When introducing watermarks, it has to be considered that a part of the error-correction capability is lost for restoring the original values in case of an error. Inducing digital watermarks can be done either by mechanically puncturing the physical media or by introducing wrong symbols in certain locations on purpose as a second data layer of watermark information. Both methods mean lowering the ability to correct errors.

It is questionable if such watermarked discs still are in accordance with the Red Book Standard (or other standards). Unless the artificially introduced error rate does not exceed the limit given by a standard, such a compact disc still can be called "CD".

This work evaluates the influence on the error correction system by executing computer simulations of the compact disc encoding and decoding procedure. Possible wa-

termark patterns have been conceived and compared with regard to their correctibility in terms of the output symbol error rate and the interpolation rate after correcting these schemes in the presence of a characteristic background noise distribution, modelling a typical user-handled disc. Optimal patterns were found and their correctibility quantitatively determined and compared to pure background noise.

Software simulations of the compact disc encoding process give a powerful means of assessing the behaviour of the error correction system. This work can easily be adapted to DVD encoding. Previously published research is based on statistical evaluation of the compact disc's decoding algorithm. That approach does not take fully into account the possibly bursty nature at the input of the C2 decoder [29]. Due to the systematic nature of additional watermarking information, either embedded in certain audio samples (symbols) or as purposefully induced defects on the actual media, an analytical statistical evaluation is not adequate. Watermarks normally consist of regular patterns.

A variety of possible patterns for watermarking a data stream by inducing purposely wrong symbols have been simulated. It was found that they differ in their error correctibility. Good patterns consist of repeating groups of maximal two symbols each new frame. It proved disadvantageous if the symbols within each frame change their position by either having random distances or by being shifted over a frame.

Error rates increase by a factor of ten if regular one-symbol errors per frame are introduced - other patterns further increase the overall error rates. In particular, randomly distributed one-symbol errors weaken the error correction almost ten times more. This is due to the likely event that two or more consecutive errors can be grouped together by the random process. Thus, it is advantageous for induced symbol errors

to be the longest distance from each other. Then the error correction (interleaving, scrambling, CIRC) is most effective. This pattern can be thinned out by placing good intermediate frames in-between frames with errors. It reduces the introduced error rate, but still imposes a minimum of burden on the error correction system compared to other more irregular patterns. It should be noted though, that due to the approximation of all patterns to $f = 1$, the differences in error rates decrease when being thinned out. Three groups of patterns can be distinguished by their speed of approximating the limit $f = 1$ (Figure 5.22, 5.23). Patterns with whole frames in error are hard to correct (EQF1/2/3). Randomly distributed symbols and equidistant symbols (RES,EQU) approach $f = 1$ faster. The optimum is achieved by regular one-symbol errors. Research in this area has not been previously published. The performance of the error correction system has only been assessed previously in the case of a random error generation.

Inducing one symbol error per frame increases the output symbol error rate by a factor of ten, as aforementioned. Since in each frame 8 symbols comprise the error correction data, using up one symbol per frame intentionally takes 12.5% of the error correction data and adds about 3% to the overall data capacity (user data + intentional watermark symbols). Thus, in the optimal case of a regular pattern, the space used for error correction data is only 20% (23% full error correction), but the error rates increase by a factor of ten, assuming a typical user-handled disc. This result shows the high capacity of the error correction system. In practice, a watermark pattern does not need that much space; using every tenth frame for one intentional symbol error is enough and would increase error rates by only approximately 1%.

Results show that the additional load on the error correction system by this optimal

pattern can be balanced by reducing the background signal noise rate by about 6%. This means in order to get the same overall error-correction rate, the limit for the maximal background noise to be allowed for (i.e. by a standard) must be reduced by 6%. This result was derived in Section 5.5.5 for input error rates of about $r = 0.02 \ldots 0.002$. The constant log-linear behaviour of both symbol error rates in Figure 5.26 suggests that it can be extrapolated to lower introduced error rates.

For the computer simulations the author assumed a characteristic distribution of background errors. The bursty nature of these background errors becomes obvious if one considers the probability density function for burst errors and the intermediate good-data gap statistics. The software incorporates this fact by emulating burst probability functions, previously published in [2], by software.

The need for considering a special type of burst distribution became obvious when comparing error-correction rates of a two-symbol error in each frame in presence of two different types of underlying noise distributions. Even with the same introduction error rate r (and the same background error rate), a two-symbol error causes different decreases of the overall error-correction rates for random noise compared to bursty noise (refer to 5.5.2). That means, in order to get reasonable results for the loss of correctibility due to watermarking, it makes sense to weight not only the error rate but also the type of background noise.

The watermark patterns presented and investigated here are only a small subset of what is possible. Especially thinning the patterns to lower the information density of a possible watermark would be practical. This in turn would increase the computing time needed to detect a reasonable number of errors. The background noise is also assumed to be relatively high in order to decrease the computing time. It is the author's belief,

though, that decreasing both, the level of background noise and the information density of watermarks, does not have an effect on the relative error frequency f in the end (refer to Section 5.5.1). That is, the diagram in Figure 5.22, 5.23 is valid for lower introduced error rates, if the background noise is reduced by the same amount.

The disadvantage of computer simulations is the extensive time needed to detect at least some errors (about 40 minimum) when the output error rate is low. Some simulation jobs were encoding and decoding 25 million frames, which give a minimum output error rate to reliably detect of about $P = 7 \cdot 10^{-8}$. Using faster computers, more time, and possibly faster algorithms this limit could be extended by at least factor 10.

This work includes also a discussion about certain error correction strategies applied in compact disc players. The author implemented two different decoders to evaluate their performance with regard to the special burst distribution mentioned earlier. One of them showed an inferior performance on both, a bursty channel and a non-bursty model.

Assessing the performance of decoder strategies with respect to a specially constructed error distribution has not been published previously. It is obvious that different kinds of error distributions cause different error-correction rates, even when the error distributions possess the same symbol error rate. It is the form and characteristic of the burst probability function that controls the performance of the error decoder.

The author's work is different from other publications by incorporating the characteristic error distribution of user-handled discs. Computer simulations in order to assess the performance of CIRC decoders in this case have been published only in [29]. The authors in [29] make use of a similar error distribution, though assuming lower error rates. Such low error rates are not practicably carried out by computer simula-

tions, therefore the author's simulations lead to significantly higher output error rates (about four orders of magnitude). This work, in particular and opposed to previous work, shows over a wide range of input error rates the difference in presupposing bursty errors of one length only [33] compared to a characteristic error distribution. The further advantage of computer simulation is the accurate modelling of the interleaver and scrambler. [29] assumes error at the input of the C2 decoder to be of a non-bursty, memoryless channel, which is a simplification, especially when dealing with regular introduced errors. Comparing the characteristic error distribution used in this work to memoryless channel errors, the output symbol error rates in the latter case amount to only 1% of the previous. Several publications assess only the non-bursty case [1, 28], therefore it was crucial to carry out further research with regard to bursts of errors.

This work dealt with computer simulations to generate characteristic error distributions and simulating the correction of them, using previously published algorithms for compact disc decoding. It was necessary to carry out this research in order to evaluate the behaviour of the error correction system in case of an introduction of watermarking schemes for compact discs. Watermarking and digital fingerprinting in general can help preventing copyright violations and help to take the control of the digital data content of a compact disc back to the publisher. It was found that digital fingerprinting methods can be implemented using either mechanical distortions in the reflective layer (secure media identifier) or marking the data stream with intentionally wrong symbols. Both methods worsen the overall error-correction capability and thus it is necessary to investigate their effects. Through computer simulations carried out in this work a quantitative evaluation was done.

It must be clarified that all watermarking technologies can only be built on a com-

plete DRM framework. Applying these techniques randomly and on its own, attacks become feasible. The success of said techniques depend also on implementing the legal necessities. Hardware manufacturers must be forced to be compliant with next generation's watermarking standards. If they do not comply, they will not be able to take part in this business.

6.2 Future Work

There are a number of issues that have not been thoroughly or not at all addressed in this work.

Digital watermarking can be applied in far more elaborate ways. It is conceivable to hide information in the control structures of the compact disc's frame layout. Control words or merging bits are only two ideas. These are usually not copyable although they could be detected easily. These measurements, as aforementioned, must all be part of a complete DRM framework.

Other watermarking schemes to evaluate encompass altering certain audio samples in the data stream in such a way that the audio compact disc player interpolates these audio samples. During copying the compact disc recorder carries out interpolation and thus introduces typical audio interpolation sequences. These sequences will stay in the data stream, even if the disc is copied many times. By searching for certain interpolated audio sequences, a compact disc player is able to detect whether a watermark is present or not. Effects comprise a decrease in sound quality. Hi-fi enthusiasts will not agree with this. Nevertheless, all watermarking techniques based on packing additional information on top of the conventional data stream lower somehow the quality of the underlying data, as shown in this work.

It is possible with the author's software to predict effects on the sound quality. Macrovision [50] uses a similar system to defy copying a compact disc by using a modified compact disc encoder, deliberately introducing potentially bad samples in certain positions, which produces spoiled audio tracks after copying [50].

Surface images captured by the author's experimental system can be processed by a technique, so-called deblurring. The deblurring algorithm takes into account the overlap of the surface laser spot when sampling adjacent points. Applying this technique, a gain in captured image quality can be expected.

Appendix A

Theory of Reed-Solomon Encoding and Decoding for Compact Discs

A.1 Galois-field Arithmetic and Basics of ECC

The mathematical properties of linear block codes and in particular BCH codes can be best understood by describing codewords with polynomials. The coefficients of the polynomials are drawn from a special set of elements, the Galois field.

Briefly, a Galois field GF is defined to have multiplication and addition form a commutative group; multiplication is distributive over addition and there are a finite number of elements in that field. Elements of a Galois field can be represented in polynomial form or in power representation.

The computation of codes therefore has to be executed using Galois field arithmetic, which the software has implemented.

Since 8-bit symbols are employed as smallest unit in the en-/decoding process, the coefficients of the polynomials are drawn from the extension field GF(2^8) of GF(2).

Elements are represented by their power form in *unsigned char*.

Multiplication is implemented in a function by using the power representation, addition needs the elements to be converted to polynomial representation and back. In order to save computer cycles during addition, an alternative, known as Zech logarithms, is employed. Defining

$$\alpha^{Z(n)} = \alpha^n + 1 \tag{A.1}$$

the sum of two elements in power representation can be written as:

$$\alpha^n + \alpha^m = \alpha^m(\alpha^{n-m} + 1) = \alpha^m \alpha^{Z(n-m)} = \alpha^{Z(n-m)+m} \tag{A.2}$$

Adding two elements is done by means of the Zech table $Z(n)$, which the constructor of the RS class generates beforehand. In order to fill the Zech table, both forms, the polynomial and the power representation, must be hold in lookup tables. The primitive polynomial

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1 \tag{A.3}$$

with coefficients from the ground field GF(2) is used to generate the extension field GF(2^8) in polynomial form according to the Standard [6].

The generator polynomial $g(x)$ of a primitive t-error correcting Reed-Solomon code of length $2^m - 1$ can be written as:

$$g(x) = (x + \alpha)(x + \alpha^2) \cdots (x + \alpha^{2t}) \tag{A.4}$$

with α being a primitive element in $GF(2^m)$.

The generator polynomial $g(x)$ is formed by the constructor of the RS class by a fast implementation with a minimum of multiplication using Galois field arithmetic.

A.2 Cyclic Code Encoding

Reed-Solomon codes form an important subclass of cyclic codes. Encoding of cyclic codes can implemented easily by employing shift registers with feedback connections. The encoding of cyclic codes in systematic form is based on the following equation:

$$\mathbf{v}(X) = X^{n-k}\mathbf{u}(X) + \mathbf{b}(X) \tag{A.5}$$

with: $\mathbf{v}(X)$: code polynomial

$\mathbf{u}(X)$: message polynomial

$\mathbf{b}(X)$: remainder of $\frac{X^{n-k}\mathbf{u}(X)}{\mathbf{g}(X)}$

$\mathbf{g}(X)$: generator polynomial

It yields to an (n, k) cyclic-code vector in systematic form consisting of $n-k$ parity check digits $(b_0, b_1, \ldots, b_{n-k-1})$ followed by k unaltered information digits $(u_0, u_1, \ldots, u_{k-1})$.

The encoding is implemented as a division circuit, which is a linear shift register with feedback connections based on the generator polynomial $\mathbf{g}(x)$ [51].

Two Reed-Solomon encoding stages, using the same set of parametrised functions, are employed, separated by a cross-interleaver in-between them. The outer decoder is a (28,24) double-error-correcting shortened Reed-Solomon Code of $d_{min} = 5$ and the inner decoder is a (32,28) double-error-correcting shortened Reed-Solomon Code of $d_{min} = 5$, both over GF(2^8) (refer to Figure 3.6).

A.3 Reed-Solomon Syndrome Decoding

This Section outlines the decoding schema employed for a Reed-Solomon Code.

The aim in decoding an error correction code is to find the most likely error pattern for a given received code word. The procedure used here can be described as syndrome decoding and is summarised for the nonbinary case as follows:

1. Calculate the syndrome values $S_k, k = 0, \ldots, 2t$.

2. Determine the error-locator polynomial $\Lambda(x)$ from the syndrome values, using the Berlekamp-Massey algorithm [52, 53], modified for erasures.

3. Solve for roots of $\Lambda(x)$, which are the error locators, using the Chien search algorithm [54].

4. Given the error locators, calculate the error values, using the Fourney-algorithm [55].

The syndromes S_k are calculated first by evaluating the received code pattern $r(x) = c(x) + e(x)$ at the roots α^k of the generator polynomial g(x), with $e(x)$ being the error pattern and $c(x)$ the original code word:

$$S_k = r(\alpha^k) = e(\alpha^k), \quad k = 0, \ldots, 2t \tag{A.6}$$

The implementation uses a convenient way to evaluate S_k:

$$S_k = \{\cdots[(r_{n-1}\alpha^k + r_{n-2})\alpha^k + r_{n-3}]\alpha^k + \cdots\}\alpha^k + r_0 \tag{A.7}$$

Introducing error magnitudes $Y_l = e_{i_l}$ and the error location numbers $X_l = \alpha^{i_l}$,

where i_l is the actual location of the lth error ($l = 1 \ldots \nu$, ν: number of errors), a set of $2t$ simultaneous equations for the syndromes can be written down:

$$S_k = Y_1 X_1^k + Y_2 X_2^k + \cdots + Y_\nu X_\nu^k, \quad k = 1, \ldots, 2t \tag{A.8}$$

This nonlinear set of equations is solved with the help of a error-location polynomial $\Lambda(x)$ which is defined as a polynomial having the roots at the inverse error locations X_l^{-1}. It leads to the following set of equations, which can be solved for the coefficients $\Lambda_1, \Lambda_2, \ldots, \Lambda_\nu$.

$$\Lambda_1 S_{j+\nu-1} + \Lambda_2 S_{j+\nu-2} + \cdots + \Lambda_\nu S_j = -S_{j+\nu}, \quad j = 1, \ldots, \nu. \tag{A.9}$$

In order to solve above equation avoiding matrix inversion, which is computationally inefficient, an algorithm, conceived independently by Berlekamp and Massey [52, 53], is employed. The algorithm attempts to build-up a linear feedback shift register with a lowest degree connection polynomial that generates the syndrome S_k. The algorithm is described in [56, 57] and implemented in the RSDecoder class with the function *berlekamp()*.

It is a slightly modified version to embrace the erasure words at the input of the decoder, giving known error locations beforehand. The theory behind it was suggested by Fourney [55]. It is based on defining an erasure-locator polynomial $\sigma'(z)$, which leads to modified syndrome values. The computation is done in the first loop of the revised berlekamp algorithm, according to [57].

Having found the coefficients of the locator polynomial $\Lambda(x)$, a method suggested by Chien [54] is applied in order to find the roots of the polynomial which give the

error location numbers X_l ([51]). The implementation is done in *chien_search()*.

Solving equation A.8 for the error magnitudes Y_l, a second matrix inversion is avoided by using Fourney's algorithm [55]. The function *correct_with_fourney()* does exactly this with additionally correcting the received code vector $r(x)$ at the know error locators X_l.

Appendix B

Software Listings

B.1 Control Software for the Experimental Apparatus

```
/*
 * adget.c
 * Copyright Kay Rydyger
 *
 * control program for SCSI player and data acquisition.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <termios.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sched.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>
#include <stdarg.h>
#include <scsi/sg.h>
#include <linux/cdrom.h>
#include <malloc.h>

/*
  compile with
  gcc  -O2 adget.c ../scsi/readinc.c ../scsi/sg_err.c -o adget
*/

#define BLOCK_SIZE 2048
```

```
#define BLOCKS 1
#ifdef SG_GET_RESERVED_SIZE
#define BLOCKS_PER_WBUFF 32    /* this implies 64 KByte working buffer */
#else
#define BLOCKS_PER_WBUFF (SG_BIG_BUFF / BLOCK_SIZE)  /* probably 32KB */
#endif

// #define SG_DEBUG

#define SG_DD_MAX_RETRIES 4

#define SG_HEAD_SZ sizeof(struct sg_header)
#define SCSI_CMD10_LEN 10
#define READ_CAP_DATA_LEN 8


#define PCXBASE 0x210
#define ADLOWBYTE PCXBASE+4
#define ADHIGHBYTE PCXBASE+5
#define DIGIIN PCXBASE+6
#define GAIN PCXBASE+9
#define CHANNEL PCXBASE+10
#define TRIGGERMODE PCXBASE+11
#define TRIGGER PCXBASE+12

#define MAXPTS 1000000
#define MAXLOOPS (MAXPTS<<1)

#define BI0 (1<<0)
#define BI1 (1<<1)
#define BI2 (1<<2)
#define BI3 (1<<3)
#define BI4 (1<<4)
#define BI5 (1<<5)
#define BI6 (1<<6)
#define BI7 (1<<7)
#define BI8 (1<<8)
#define BI9 (1<<9)
#define BI10 (1<<10)
#define BI11 (1<<11)
#define BI12 (1<<12)
#define BI13 (1<<13)
#define BI14 (1<<14)
#define BI15 (1<<15)

//#define DEBUG (BI3|BI4|BI6|BI9|BI12)
#define LOG(loglevel,a) if ( loglevel & debuglevel ) if (timestamps) {printf("%lf: ",get_us()/1e6); printf a; f


static int pid,status,fdbin;
static int endmark=123456789, tracks, interval=2; // 2 microseconds
static char digi[MAXLOOPS], accuData[MAXLOOPS];
static FILE *fp;
static struct timespec time_to_sleep;
static realtime_sched=0, usescsicmd=0, audiocd=0, revolutions=20000;
static int starting_track=1,ending_track=1,timestamps=0,debuglevel=8191;
static int dontstartshell=0;
FILE *fpECLog;
int fdlog;

extern char **environ;

double get_us(void)
{
        struct timeval tv;
        struct timezone tz;
        tz.tz_minuteswest=5;
        gettimeofday(&tv,&tz);
        return (double)tv.tv_sec * 1e6 + tv.tv_usec;
}

static inline void outb(short port, char val)
{
```

```
        ioperm(port, 1, 1);
        __asm__ volatile ("out%B0 %0,%1" : :"a" (val), "d" (port));
}

static inline unsigned char inb(short port)
{
        unsigned int ret;
        ioperm(port, 1, 1);
        __asm__ volatile ("in%B0 %1,%0" : "=a" (ret) : "d" (port));
        return ret;
}

void init_AD(void)
{
        outb( TRIGGERMODE, 1 ); /* enable software trigger */
        outb( GAIN, 2 );  /* 2: set gain to   +/-2.5V  kein 1.25V! durch Ausprobieren gefunden */
        outb( CHANNEL, 0 ); /* set channel to 0 */
}

/*
void sg_start_unit(int fd)
{}
*/

void  wait_for_trigger(void)
{
        int h;
        while ( (h=inb( DIGIIN )&(char)1) == 1 );
        while ( (h=inb( DIGIIN )&1) == 0 );
}

int start_accumulation(double *time)
{
        int counter,h, sectors;
        char *digiptr = digi;
        double time_used_us, start1, stop1, used_us_per_count;

        counter=0;
        start1 = get_us();
        LOG(BI3,("- starting capturing - revolutions: %d\n",tracks));
        while ( counter < MAXPTS )
        {
                counter++;
                h = inb(DIGIIN);
                if ( (h&(char)1) == 0 ) break;
                outb(TRIGGER,0);
/*
  LOW byte first if we use perl's "unpack s,.."
  define SWAP_BYTES in X.c, for this case
*/
                *digiptr++ = inb(ADLOWBYTE);
                *digiptr++ = inb(ADHIGHBYTE);
                    /*
                h = inb(DIGIIN);
                if ( (h&(char)1) == 0 ) break;
                    */
                if (realtime_sched )
                        nanosleep(&time_to_sleep,NULL);
        }
        stop1 = get_us();
        time_used_us = (stop1 - start1);
        LOG(BI3, ("--- stopped capturing - time %lf s, number of samples %d\n",
                  time_used_us/1e6, counter));
        if ( counter >= MAXPTS )
        {
                LOG(BI6,("CD stopped!\n"));
                write( fdbin, &endmark, sizeof(int) );
                exit(2);
        }
        if (counter != 0)
                used_us_per_count = time_used_us / counter;
        else {
                used_us_per_count=0;
```

```
                time_used_us=0;
        }
        LOG(BI3,("average time per sample interval %lf us\n",used_us_per_count));
        sectors = time_used_us * 75 / 1e6;
        LOG(BI5,("Sectors : %d\n",sectors));
//        fflush(stdout);
        *time = time_used_us;
        return counter<<1;
            /*   return number of bytes  */
}

void binary_save(char *begin_buf, int count)
{
        write(fdbin,&count,sizeof(int));
        write(fdbin,begin_buf,count);
}

double average(double med, double new)
{
        static int N=0, N1=1;
        N++;N1++;
        return N/(float)N1 * med + 1/(float)N1 * new;
}


int main_acquisition_loop(int revs)
{
        int c, rev_lost;
        double med_time_used, time_ges=0, time_used;
        while ( tracks++ < revs )
        {
                wait_for_trigger();
                c = start_accumulation(&time_used);

                if ( time_used != 0 )
                {
                        if ( tracks==1 ) med_time_used = time_used;
                        if ( time_used > med_time_used * 1.7 )
                        {
                                int i;
                                rev_lost = (int) (time_used/med_time_used +.5);
                                LOG(BI9,("sync lost %d times\n",rev_lost));
                                tracks += (rev_lost-1);
                                for (i=0; i<rev_lost; i++)
                                        binary_save(digi+c/rev_lost*i,c/rev_lost);
//                                          binary_save(digi+(c>>1),c>>1);

                        }
                        else
                        {
                                med_time_used=average(med_time_used,time_used);
                                binary_save(digi,c);
                        }
                        time_ges += time_used;
                }

                else tracks--;

//                LOG(BI3,("average time: %lf s\n",med_time_used/1e6));


        }
        return time_ges;
}

void save_data(int c)
{
        int i=0, value;
        LOG(BI3,("..saving data\n"));
        while (i<c)
        {
                value =   digi[i];     i++;
                value += (digi[i]<<8); i++;
```

```
                fprintf(fp,"%d\n",value);
        }
}

void interrupt_handler(int sig)
{
        LOG(BI9,("\nInterrupt Signal caught at track %d .. exiting!\n\n",tracks));
        write( fdbin, &endmark, sizeof(int) );

        kill(pid,9);  /* die, son ! */
        wait(&status);
        fclose(fp);
        close(fdbin);
//        exit (0);
}

void sigchild_handler(int sig)
{
        LOG(BI9,("pid: %d, child died !\n\n exiting at track %d\n",pid,tracks));
        write( fdbin, &endmark, sizeof(int) );

        wait(&status);
        fclose(fp);
        close(fdbin);
        exit (0);
}

void display_usage(void)
{
        printf("adget [-o outputfile | -b # | -s | -r | -a | -v # | -l logfile | -p # | -t # #| -D device | -L
        printf(" -s : scsi commands, -r : realtime scheduling, -a : audio CD, -b : begin_block, -v : revolution
        printf(" default: -o adbin.dat -v 20000 -p 2 -t 1 1 -l logAD.txt -D /dev/sg0 -b 0 -L 8192\n");
}

int main(int argc, char **argv)
{
        int i,interval_set=0, starting_set=0;
        int begin_block=0;
        double time_ges, program_start_time, program_stop_time;
        struct sched_param sched;
        char Outputf[32]="adbin.dat", logfile[32]="logAD.txt";
        char Device[32]="/dev/sg0";
        struct stat statbuf;

        for(i=1;i<argc;i++)
        {
                if (argv[i][0]=='-')
                {
                        switch (argv[i][1]) {
                            case 'D':
                                    if (++i<argc)
                                            strncpy(Device, argv[i],32);
                                    else
                                    {
                                            printf("wrong argument  near \"-D\"\n\n");
                                            exit(1);
                                    }
                                    break;
                            case 'o':
                                    if (++i<argc )
                                            strncpy(Outputf, argv[i],32);
                                    else
                                    {
                                            printf("wrong argument  near \"-o\"\n\n");
                                            exit(1);
                                    }
                                    break;
                            case 'l':
                                    if (++i<argc )
                                            strncpy(logfile, argv[i],32);
                                    else
                                    {
                                            printf("wrong argument  near \"-l\"\n\n");
```

```
                        exit(1);
                }
                break;
        case 's': usescsicmd=1;
                break;
        case 'a': audiocd=1;
                break;
        case 'r': realtime_sched=1;
                break;
        case 'T': timestamps=1;
                break;
        case 'n': dontstartshell=1;
                break;
        case 'h': display_usage();
                exit(0);
        case 'p':
                if (++i<argc )
                {
                        interval = atoi(argv[i]);
                        interval_set=1;
                }

                else
                {
                        printf("wrong argument  near \"-p\"\n\n");
                        exit(1);
                }
                break;
        case 'b':
                if (++i<argc )
                        begin_block = atoi(argv[i]);
                else
                {
                        printf("wrong argument  near \"-b\"\n\n");
                        exit(1);
                }
                break;
        case 'v':
                if (++i<argc )
                        revolutions = atoi(argv[i]);
                else
                {
                        printf("wrong argument  near \"-v\"\n\n");
                        exit(1);
                }
                break;
        case 't':
                if (++i<argc-1 && argv[i+1][0]!='-')
                {
                        starting_set=1;
                        starting_track = atoi(argv[i]);
                        ending_track = atoi(argv[++i]);
                }
                else
                {
                        printf("wrong argument  near \"-t\"\n\n");
                        exit(1);
                }
                break;
        case 'L':
                if (++i<argc )
                        debuglevel = atoi(argv[i]);
                else
                {
                        printf("wrong argument  near \"-L\"\n\n");
                        exit(1);
                }
                break;
        default: printf("wrong argument near %s\n\n",argv[i]);
                exit(1);
        }

} else {
```

```
                printf("wrong argument near %s\n\n",argv[i]);
                exit(1);
        }
}

if (interval_set && !realtime_sched)
{
        printf("setting interval without realtime scheduling doesn't make sense!\n\n");
        exit(1);
}

if ( starting_set && !audiocd && !usescsicmd )
{
        printf("setting starting and end track requires either -a or -s\n");
        exit(1);
}

if (( fp=fopen("ad.dat","w")) == NULL ) {
        perror("fopen");
        exit(1);
}

if (( fdbin = creat(Outputf,0777) ) == -1)
{
        perror("creat");
        exit(1);
}

    // fchown(fd,1000,1000);
signal(SIGINT,interrupt_handler);
signal(SIGCHLD, sigchild_handler);

init_AD();

if ( realtime_sched )
{
        time_to_sleep.tv_sec=0;
        time_to_sleep.tv_nsec=interval*1000;
        sched.sched_priority = 30;
        sched_setscheduler(0, SCHED_RR, &sched);
        LOG(BI6,("\nrunning process changed to SCHED_RR scheduling!\n\n"));
        if (!dontstartshell) {
                if ( (pid = fork()) < 0 )
                {
                        perror("fork");
                        exit(1);
                }
                if (pid==0)  // Sohn!
                {
                        char *argv[4];
                        printf("starting shell\n\n");
                        sched.sched_priority = 95;
                        sched_setscheduler(0, SCHED_RR, &sched);
                        system("/bin/bash");
                        exit(0);
                }
        }
        fdlog = open(logfile, O_RDWR | O_CREAT | O_TRUNC);
        if ( fdlog < 0 ) LOG(BI12,("error opening logfile %s,%d\n",logfile,fdlog));
        if ( dup2(fdlog,1 ) < 0 ) LOG(BI12,("error duplicating fd %d to 1\n",fdlog));
        if ( dup2(fdlog,2 ) < 0 ) LOG(BI12,("error duplicating fd %d to 2\n",fdlog));
        if ( close(fdlog) < 0 )
                LOG(BI12,("error closing logfile %s,fd %d\n",logfile,fdlog));
}



tracks=0;
if ( !audiocd )
        if ( (pid = fork()) < 0 )
        {
                perror("fork");
                exit(1);
```

```
                }


        if ( !audiocd && pid==0 ) /* Sohn */ // Data CD section
        {
                if ( usescsicmd )
                {
                        int fd,n,ret;

                        char EClogf[32]="EClog.dat";

                        unsigned char *wrkBuff= malloc(SG_HEAD_SZ + SCSI_CMD10_LEN +
                                                        (BLOCK_SIZE * BLOCKS));
                        fd = open(Device, O_RDWR);
                        if (fd < 0)
                        {
                                perror("open");
                                exit(1);
                        }

                        fpEClog = fopen(EClogf,"w");
                        if ( fpEClog == NULL )
                        {
                                LOG(BI12,("error in fopen %s\n",argv[2]));
                                perror("fopen");
                                exit(1);
                        }

                        if (0 == wrkBuff) {
                                LOG(BI12,("can't alloc mem for wrkBuf\n"));
                                perror("malloc");
                                exit(1);
                        }

                        for (n=begin_block;;n+=BLOCKS)
                        {
                                LOG(BI4,(" -- accessing sector %d\n",n));
//                                fflush(fpEClog); fflush(stdout);
                                ret = sg_read(fd,wrkBuff,BLOCKS,n);

                        }
                        fclose(fpEClog);
                        exit(2);

                }

                else {

/* no SCSI read */
                        char buf[2048];
                        int fd, ret,sector;
                        if ( (fd = open( Device, O_RDONLY | O_NONBLOCK )) < 0 ) {
                                perror("scsi_open");
                                exit(1);
                        }
//                lseek( fd, 1100000, SEEK_SET );

                        for (sector=0;;sector++) {
                                ret = read(fd,buf,sizeof(buf));
                                if (ret < 0) {
                                        LOG(BI12,("read error %s\nskipping...\n",
                                                strerror(errno)));
                                        lseek(fd,2048*200,SEEK_CUR);
                                }

                        }

                }

        }
        if ( audiocd ) // Audio CD section
        {
                int fd,ret;
```

```
                char EClogf[32]="EClog.dat";

                unsigned char *wrkBuff= malloc(SG_HEAD_SZ + SCSI_CMD10_LEN +
                                               (BLOCK_SIZE * BLOCKS));
                fd = open(Device, O_RDWR);
                if (fd < 0)
                {
                        perror("open");
                        exit(1);
                }

                fpEClog = fopen(EClogf,"w");
                if ( fpEClog == NULL )
                {
                        LOG(BI12,("error in fopen %s\n",argv[2]));
                        perror("fopen");
                        exit(1);
                }

                if (0 == wrkBuff) {
                        LOG(BI12,("can't alloc mem for wrkBuf\n"));
                        perror("malloc");
                        return 1;
                }

                if ( usescsicmd )
                {
//                        sg_start_unit(fd,wrkBuff);
                        ret = sg_play_audio(fd,wrkBuff,starting_track,ending_track);
                }

                else
                {
                        LOG(BI9,("warning: not using scsi commands, CD player must be started with independant
                }

        }

        program_start_time = get_us();

        time_ges = main_acquisition_loop(revolutions);
        write( fdbin, &endmark, sizeof(int) );

        program_stop_time = get_us();

        LOG(BI3,("average time per count: %f s\n",time_ges/tracks/1e6));
        LOG(BI3,("total time used:        %f s\n\n",
                 (program_stop_time-program_start_time)/1e6));

        close(fdbin);
        fclose(fp);

        if ( kill(pid,9) < 0 )
        {
                LOG(BI12,("kill error pid %d\n",pid));
                perror("kill");
        }
            /* not reached ! */
        exit(0);
}
```

B.2 Simulation Software for Compact Disc Channel Modelling

B.2.1 Flowchart of the Encoder and Decoder Program

Figure B.1 and B.2 show a schematic view of the workflow of both the encoder and decoder. The main methods for each class, as well as each class is listed and classified by where the functions are called from and whether they are public or protected class members. Inheritance between classes is shown as well. A short description about every class is included. All classes and functions can be found in the following listing (B.2.2). Further information, in particular about Reed-Solomon encoding and decoding, is given in Appendix A, in Section 3.3 the software implementation is discussed.

B.2.2 Program listing

```
/*
 * main.cc
 * Copyright Kay Rydyger
 *
 * main program for channel simulation
 */

#include "defs.h"
#include "classes.h"

#include <unistd.h>
#include <time.h>

static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

#define MINBURSTFR 13
#define MAXBURSTFR 250
#include "patterns.h"

int frame, MaxFrame=3000;
static int gaps=0, bursts=0, gap_GPC_wide=3, burst_GPC_wide=3;
static int frame_offset=0;
static void makeFramebad(int*, int*);
static int orgFrame[FC1SZBCW];
static int frpat[100];
static int modified_symbols=0, not_modified_symbols=0;
static int modified_symbols_PDF=0, not_modified_symbols_PDF=0;
static int intro_symbols=0, ok_symbols=0;
static int overall_errors=0, overall_not_errors=0;
static float BER=0.001, thinning=1; // default BER, thinning
static time_t starttime;

void introducePDFBursts(int *Frame, Random *rnd, float thinning)
```

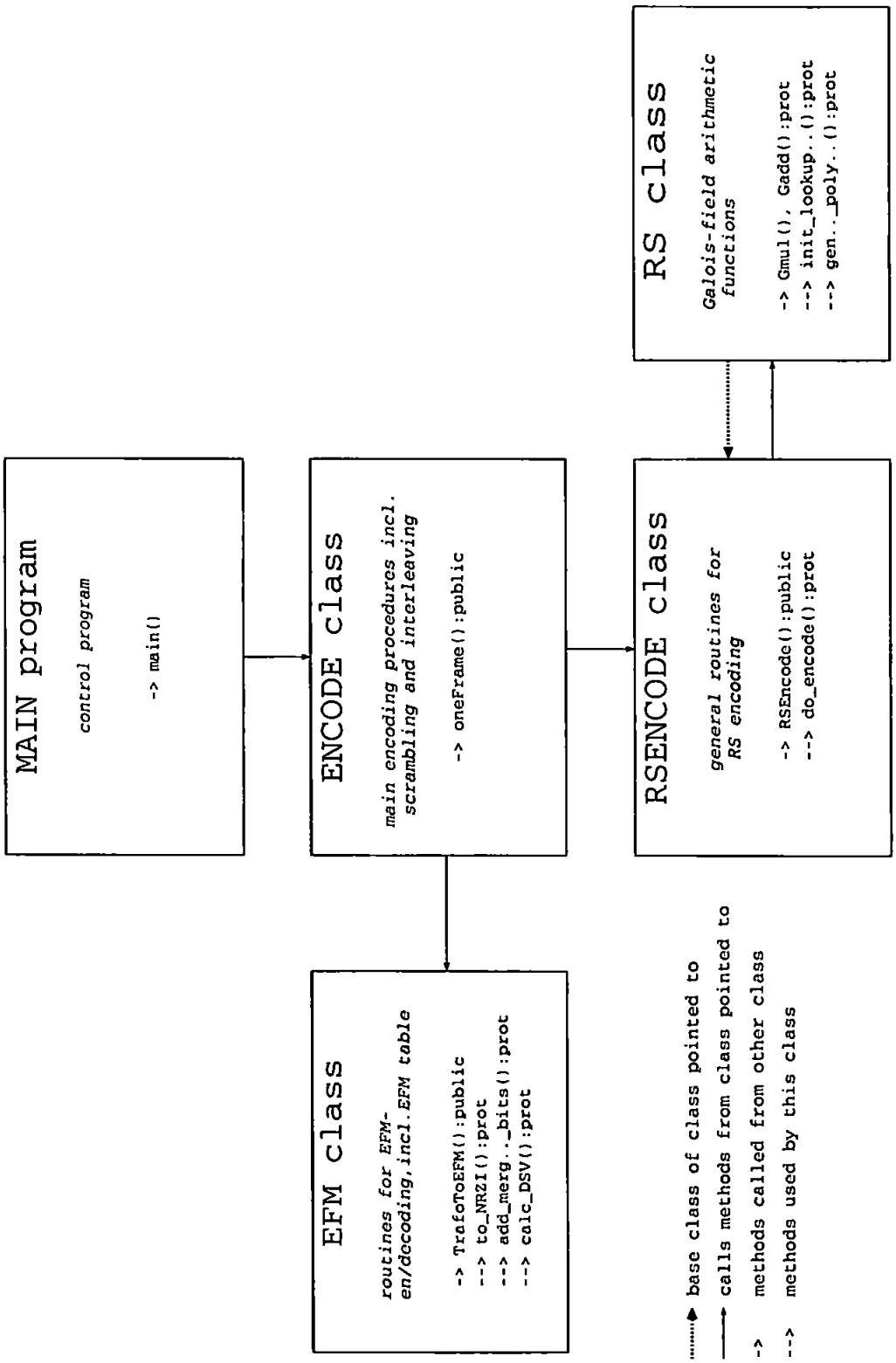


Figure B.1: Flowchart of the channel encoder implementation for CD audio.

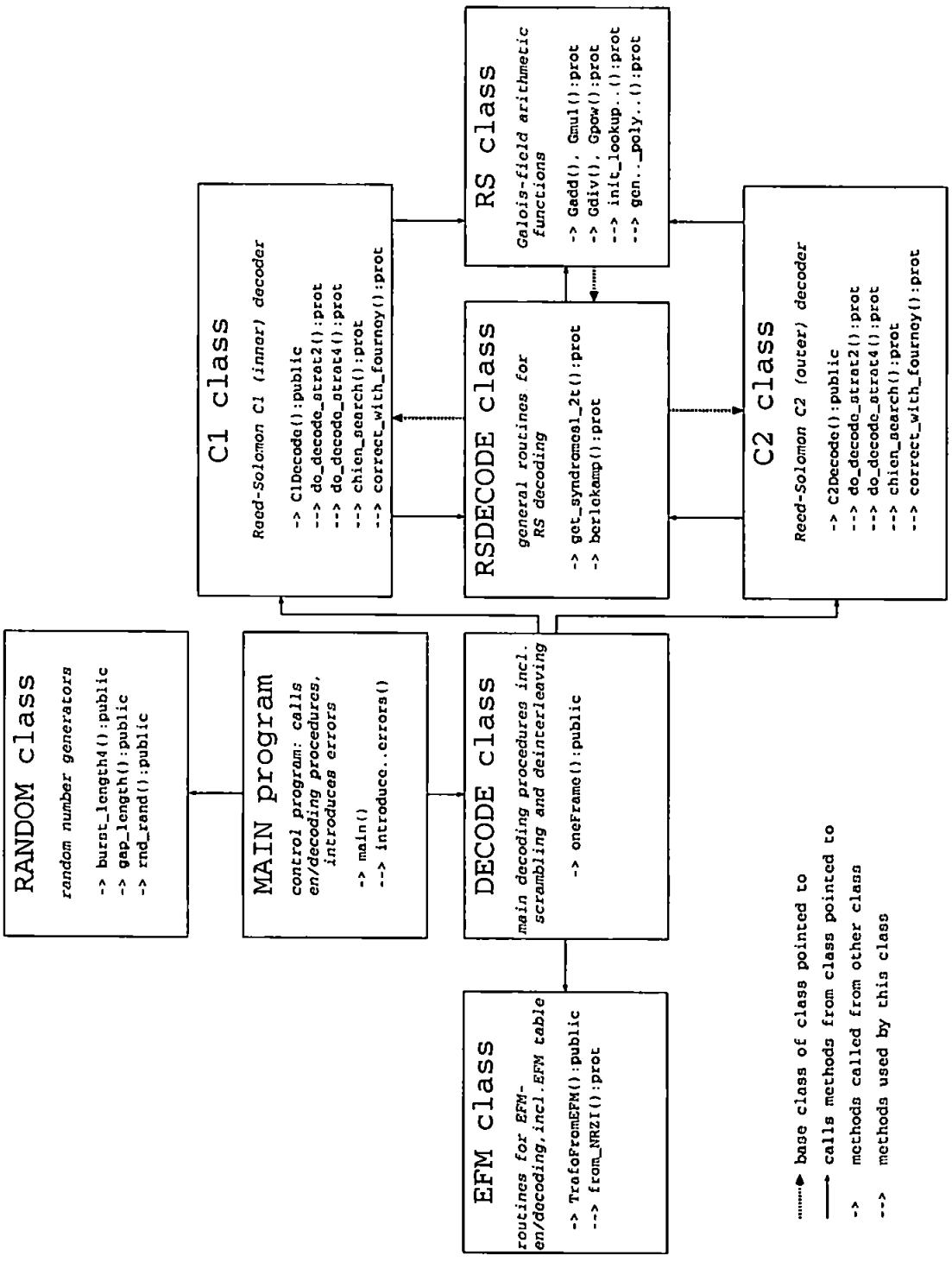


Figure B.2: Flowchart of the channel decoder implementation for CD audio.

```
{
  int i;
  static int gap_cnt = rnd->gap_rnd_length(thinning), burst_cnt = 0;
  static int burst = 0, gap = 1;

  for (i=0;i<FC1SZBCW;i++)
    /*
      sync bits not chosen!
      CW counted because possible scratches have effect on CW, too
      3% difference
    */
    {
      if (burst)
        {
          modified_symbols_PDF++;
          burst_cnt--;
          Frame[i] = rnd->rnd_rand(maxft14);
          if (!burst_cnt)
            {
              gap_cnt = rnd->gap_rnd_length3(thinning);
              gaps++; gap=1; burst=0;
            }
        }
      else if (gap)
        {
          not_modified_symbols_PDF++;
          gap_cnt--;
          if (!gap_cnt)
            {
              burst_cnt = rnd->burst_rnd_length4();
              burst=1; gap=0; bursts++;
            }
        }

    }
}

void introduceEquidistantErrors(Random *rnd, Decode *de, int distance)
{
  /* controlled by option -g (gap_GPC_wide) <-> do not use with introduceGPCErrors() */
  int i;
  static int z=0;

  if (frame>120)
    for (i=1;i<FC1SZBCW;i++,z++)
        if ( z % distance == 0)
        {
          de->inputFrame[i] = 0;
          intro_symbols++;
          z=0;
        }
      else ok_symbols++;
}

void introduceErrorSequence(Random *rnd,Decode *de, int many, int fr_off)
{
  /* controlled by option -g (gap_GPC_wide) <-> do not use with introduceGPCErrors() */
  int i;
  if (frame>120)
    for (i=1;i<FC1SZBCW;i++)
      if (i >= fr_off+1 && i < fr_off+many+1)
        {
          de->inputFrame[i] = 0;
          intro_symbols++;
        }
      else ok_symbols++;
}

void introduceErrorSequenceShifted(Random *rnd,Decode *de, int many)
{
  /* controlled by option -g (gap_GPC_wide) <-> do not use with introduceGPCErrors() */
  static int shift=0;
  int i;
```

```
  if (frame>120)
    {
      if (shift>(FC1SZB-many)) shift=0;
      for (i=1;i<FC1SZBCW;i++)
        if (i >= shift && i < shift+many)
          {
            de->inputFrame[i] = 0;
            intro_symbols++;
          }
        else ok_symbols++;
      shift++;
    }
}

void introduceErrorGroups(Random *rnd,Decode *de, int many)
{
  /* controlled by option -g (gap_GPC_wide) <-> do not use with introduceGPCErrors() */
  int z=0, i, v=30/(int)((many-2)/(float)2+1.5)+1;

  if (frame>120)
    {
      for (i=1;i<FC1SZBCW;i++)
        if ((i%v==0 || i%v==1) && z<many)
          {
            de->inputFrame[i] = 0;
            intro_symbols++;
            z++;
          }
        else ok_symbols++;
    }
}

void introduceRandomErrorSymbols(Random *rnd, Decode *de, int width)
{
  int i;
  static int gap_cnt=0;
  if (frame>120)
    {
      for (i=0;i<FC1SZB;i++)
        {
          if (!gap_cnt--)
            {
              de->inputFrame[i] = 0;
              intro_symbols++;
              gap_cnt = rnd->rnd_randfast(width);
            }
          else ok_symbols++;
        }
    }
}

void introduceGPCErrors(int *Frame, GPC *gpc, Random *rnd)
{
  int i, wide=2;
  static int gap_cnt = gpc->gap_gpc_length(rnd,gap_GPC_wide), burst_cnt = 0;
  static int burst = 0, gap = 1;

  for (i=0;i<FC1SZBCW;i++)   // sync bits not chosen!
    {
      if (burst)
        {
          modified_symbols++;
          burst_cnt--;
          Frame[i]-=3;
          if (!burst_cnt)
            {
              gap_cnt = gpc->gap_gpc_length(rnd,gap_GPC_wide);
              if (gap_cnt==0) gap_cnt++;
              gap=1; burst=0;
              printf("GPC:frame %d  gap length %d nr of frames %d\n",
                     frame,gap_cnt,gap_cnt/33+1);
              continue;
            }
```

```
        }
      if (gap)
        {
          not_modified_symbols++;
          gap_cnt--;
          if (!gap_cnt)
            {
              burst_cnt = gpc->burst_gpc_length(rnd,burst_GPC_wide);
              burst=1; gap=0;
              printf("GPC:frame %d  burst length %d nr of frames %d\n",
                     frame,burst_cnt,burst_cnt/33+1);
            }
        }
    }
}

void introduceRandomErrors(Random *rnd, Decode *de, int random_part)
{
  int i,zufall;

  if (frame>120)
    for (i=0;i<FC1SZB;i++)
      if ( rnd->rnd_rand(BERFACTOR) < random_part) {
        de->inputFrame[i] = 0;
        modified_symbols++;
      }
      else not_modified_symbols++;
}

void introduceDoubleRandomErrors(Random *rnd, Decode *de, int random_part)
{
  int i;
  if (frame>120)
    for (i=0;i<FC1SZB;i++)
      if ( rnd->rnd_rand(BERFACTOR) < random_part) {
        de->inputFrame[i] = rnd->rnd_rand(maxft14);
        modified_symbols++;
        if (i<FC1SZB-1) {
          de->inputFrame[++i] = rnd->rnd_rand(maxft14);
          modified_symbols++;
        }
      }
      else not_modified_symbols++;
}

void introduceTripleRandomErrors(Random *rnd, Decode *de, int random_part)
{
  int i;
  if (frame>120)
    for (i=0;i<FC1SZB;i++)
      if ( rnd->rnd_rand(BERFACTOR) < random_part) {
        de->inputFrame[i] = rnd->rnd_rand(maxft14);
        modified_symbols++;
        if (i<FC1SZB-1) {
          de->inputFrame[++i] = rnd->rnd_rand(maxft14);
          modified_symbols++;
        }
        if (i<FC1SZB-1) {
          de->inputFrame[++i] = rnd->rnd_rand(maxft14);
          modified_symbols++;
        }
      }
      else not_modified_symbols++;
}

void introduceFrameError(int *Frame,int *pat, int distance)
{
  int i;
  if (frame%(distance+1)==0)
    {
      for (i=0;i<FC1SZBCW;i++)
        if (pat[i])
          {
```

```
            Frame[i]=~FILLBYTE;
            intro_symbols++;
          }
        else ok_symbols++;
    }
  else ok_symbols+=FC1SZBCW;
}

void accumulate_overall_errors(int *copy,int *orig)
{
  int i;
  if (frame>120)
    for (i=0;i<FC1SZB;i++)
      if (copy[i]!=orig[i])
        overall_errors++;
      else
        overall_not_errors++;
}

void makeFramebad(int *Frame,int *pat)
{
  int i;
  for (i=0;i<FC1SZBCW;i++)
    if (pat[i])
      {
        Frame[i]=~FILLBYTE;
        intro_symbols++;
      }
    else ok_symbols++;
}
void introduceOneBurstError(int *Frame, int framestart, int frameend, int *pat)
{
        if (frame >= framestart && frame < frameend)
                makeFramebad(Frame,pat);

}

void setFramecontents(union Frame *in,char fill)
{
        int j;
        for (j=0;j<sizeof(*in);j++)
                in->byte[j] = fill;
}

void testmaxBurst(int pnu, int strat)
{
        int burstframes;
        Decode *De;

        for (burstframes=MINBURSTFR; burstframes<MAXBURSTFR; burstframes++)
        {
                INFO(4,("testing burst of %d frames\n",burstframes));
                De = new Decode(strat);
                for (frame=0; frame<MaxFrame; frame++)
                {
                        memcpy(De->inputFrame,orgFrame,FC1SZBCW*sizeof(int));
                        introduceOneBurstError(De->inputFrame, 200, 200+burstframes, pattern[pnu]);
                        De->oneFrame();
                        if ( frame>109 && De->modified() )
                          {
                            printf("testmaxBurst: error detected in frame %d:  max. burstlength = %d frames in
                                return;
                        }
                }
                delete De;
        }
        if ( burstframes == MAXBURSTFR ) printf("burstframes=%d! might be longer..\n",MAXBURSTFR);
}

void testFramepattern(int strat)
{
        int burstfr=0, i=0;
        Decode *De;
```

```
        De = new Decode(strat);

        for ( frame=0; frame<MaxFrame; frame++)
        {
                memcpy(De->inputFrame,orgFrame,FC1SZBCW*sizeof(int));
                if ( frame >= 115 )
                {
                        if ( burstfr == frpat[i] )
                        {
                                i++; burstfr=0;
                        }
                        burstfr++;
                        if ( i%2==0 && frpat[i]!=0 )
                                makeFramebad(De->inputFrame,pattern[0]);
                }
                De->oneFrame();
                if ( frame >= 115 && De->modified() )
                  {
                    printf("testFramepattern: error detected in frame %d, i=%d, burstfr=%d\n",frame,i,burstfr);
                        return;
                }
        }
        delete De;
}

void check_decoding_frame_result(BYTE* framebytes,
                                 BYTE* erabytes,
                                 struct Error_kinds *errs)
{
  int i, left;
  static int symb_counter_dec=240;
  static int eras_in_seq[2]={0,0}, errs_not_eras_in_seq[2]={0,0},
                                     errs_in_seq[2]={0,0};

  if (frame>120)
    for (i=0;i<FSZB;i++,symb_counter_dec++)
      {
        left=i%2;
        errs->bytes_checked++;
        if (erabytes[i]) {
          eras_in_seq[left]++;
          if (framebytes[i] != (unsigned char) symb_counter_dec) errs->P11++;
          else errs->P01++;
        }
        else {
          if (eras_in_seq[left]==1||eras_in_seq[left]==2) {
            errs->interpolated_eras++;
          }
          eras_in_seq[left]=0;
          if (framebytes[i] != (unsigned char) symb_counter_dec) errs->P10++;
          else errs->P00++;
        }
        if (framebytes[i] != (unsigned char) symb_counter_dec) {
          INFO(1,("byte %d modified in frame %d\n",i,frame));
          errs->single_errors++;
          errs_in_seq[left]++;
        }
        else
          if (errs_in_seq[left]) {
            errs->error_clicks++;
            errs_in_seq[left]=0;
          }
        if ( framebytes[i] != (unsigned char) symb_counter_dec &&
             !erabytes[i] ) // miscorrection ??
          errs_not_eras_in_seq[left]++;
        else
          if (errs_not_eras_in_seq[left]) {
            errs->errs_not_eras++;
            errs_not_eras_in_seq[left]=0;
          }
      }
}
```

```
time_t print_current_time(char* message, time_t starttime)
{
  time_t timep;
  timep = time(NULL);
  printf("Process ID %d %s %s",getpid(),message,ctime(&timep));
  if (starttime) printf("Process ID %d used time is: %d min %d s\n",
                        getpid(), (timep-starttime)/60,(timep-starttime)%60);
  fflush(stdout);
  return timep;
}

void print_parameters(char **argv)
{
  int i;
  printf("called with: ");
  for (i=0;argv[i]!=NULL;i++)
    printf("%s ",argv[i]);
  putchar('\n');
}

void print_mode(int Mode)
{
  if (Mode&1)
    {
      printf("Mode RAN chosen\n");
      printf("BER is %f (option -B)\n",BER);
    }
  if (Mode&2)
    {
      printf("Mode PDF chosen\n");
      printf("thinning is %f (option -n)\n",thinning);
    }
  if (Mode&4)
    {
      printf("Mode GPC chosen\n");
      printf("distance is %d symbols (option -g, gap_GPC_wide)\n",gap_GPC_wide);
    }
  if (Mode&8)
    {
      printf("Mode DRN chosen\n");
      printf("BER is %f (option -B)\n",BER);
    }
  if (Mode&16)
    {
      printf("Mode TRN chosen\n");
      printf("BER is %f (option -B)\n",BER);
    }
  if (Mode&32)
    {
      printf("Mode EQU chosen, ");
      printf("distance is %d symbols (option -g, gap_GPC_wide)\n",gap_GPC_wide);
    }
  if (Mode&64)
    {
      printf("Mode ESE chosen, ");
      printf("sequence of %d symbols (option -g, gap_GPC_wide), frame_offset is %d symbols\n",gap_GPC_wide,fram
    }
  if (Mode&128)
    {
      printf("Mode EQF1 chosen, ");
      printf("distance is %d frames (option -g, gap_GPC_wide)\n",gap_GPC_wide);
    }
  if (Mode&256)
    {
      printf("Mode EQF2 chosen, ");
      printf("distance is %d frames (option -g, gap_GPC_wide)\n",gap_GPC_wide);
    }
  if (Mode&512)
    {
      printf("Mode ESS chosen, ");
      printf("sequence of %d symbols (option -g, gap_GPC_wide)\n",gap_GPC_wide);
    }
```

```
  if (Mode&1024)
    {
      printf("Mode EGR chosen, ");
      printf("%d symbols per frame (option -g, gap_GPC_wide)\n",gap_GPC_wide);
    }
  if (Mode&2048)
    {
      printf("Mode EQF3 chosen, ");
      printf("distance is %d frames (option -g, gap_GPC_wide)\n",gap_GPC_wide);
    }
  if (Mode&4096)
    {
      printf("Mode RES chosen, ");
      printf("width is %d symbols (option -g, gap_GPC_wide)\n",gap_GPC_wide);
    }
}

void init_log(FILE **flog)
{
  char logfile[32];
  sprintf(logfile,"run-%d.log",getpid());
  if (chdir("./log")!=0)
    {
      perror("no log dir!");
      exit(1);
    }
  *flog = fopen(logfile,"w+");
  if (*flog==NULL)
    {
      perror("run in logframes():");
      exit(1);
    }
}

void print_encoded_frame(Decode *de, int* framebytes)
{
  int i;
  if (frame<=120) return;
  for (i=0;i<FC1SZB;i++)
    printf(" %2X",de->Efm->TrafoFromEFM(framebytes[i]));
  printf("\n");
}

int main(int argc, char **argv)
{
  unsigned j;
  int i=0, pnu, optburst=0,optframe=0,optspread=0;
  int encodeonly=0, decodeonly=0, maxframes=1000, encode_decode=0; // maxframes=1000
  int length=200, logframes=0; // default length=200
  int Strategy_nu=4, Mode=0, distro=0;
  FILE *flog;
  char filename[255];

  Encode *En;
  Decode *De;
  Random *Rnd;
  GPC *Gpc;

  if ( argc<2)
    {
      printf("Usage: run [options]\n options are:\n -t <maxframes> - Decoding -> Decoding\n -s <length> - get
      exit(1);
    }

  for(i=1;i<argc;i++)
    {
      if (argv[i][0]=='-')
        {
          switch (argv[i][1]) {
          case 't': encode_decode=1;
            if (++i>=argc||(maxframes = atoi(argv[i]))==0)
              {
                printf("wrong argument near \"-t\"\n\n");
```

```
      exit(1);
    }
  break;
case 'b': optburst=1;
  break;
case 'e': encodeonly=1;
  if (++i>=argc)
    {
      printf("wrong argument near \"-e\"\n\n");
      exit(1);
    }
  else strncpy(filename,argv[i],255);
  break;
case 'd': decodeonly=1;
  if (++i>=argc)
    {
      printf("wrong argument near \"-d\"\n\n");
      exit(1);
    }
  else strncpy(filename,argv[i],255);
  break;
case 'f': optframe=1;
  for(j=i+1;j<argc&&argv[j][0]!='-';j++)
    frpat[j-i-1] = atoi(argv[j]);
  frpat[j-i-1]=0;i=j-1;
  if (frpat[0]==0) {
    printf("wrong argument near \"-f\"\n\n");
    exit(1);
  }
  break;
case 's': optspread=1;
  if (++i>=argc||(length = atoi(argv[i]))==0)
    {
      printf("wrong argument near \"-s\"\n\n");
      exit(1);
    }
  break;
case 'n':
  if (++i>=argc||(thinning = atof(argv[i]))==0)
    {
      printf("wrong argument near \"-n\"\n\n");
      exit(1);
    }
  break;
case 'm':
  if (++i>=argc||(MaxFrame=atoi(argv[i]))==0)
    {
      printf("wrong argument near \"-m\"\n\n");
      exit(1);
    }
  break;
case 'B':
  if (++i>=argc||(BER=atof(argv[i]))==0)
    {
      printf("wrong argument near \"-B\"\n\n");
      exit(1);
    }
  break;
case 'g':
  if (++i>=argc)
    {
      printf("wrong argument near \"-g\"\n\n");
      exit(1);
    }
  else gap_GPC_wide = atoi(argv[i]);
  break;
case 'w':
  if (++i>=argc||(burst_GPC_wide = atoi(argv[i]))==0)
    {
      printf("wrong argument near \"-w\"\n\n");
      exit(1);
    }
  break;
```

```
        case 'L': logframes=1;
          break;
        case 'v': distro=1;
          break;
        case 'S':
          if (++i>=argc||(Strategy_nu = atoi(argv[i]))==0)
            {
              printf("wrong argument near \"-S\"\n\n");
              exit(1);
            }
          break;
        case 'M':
          if (++i>=argc)
            {
              printf("wrong argument near \"-M\"\n\n");
              exit(1);
            }
          Mode = atoi(argv[i]);
          break;
        case 'o':
          if (++i>=argc)
            {
              printf("wrong argument near \"-o\"\n\n");
              exit(1);
            }
          frame_offset = atoi(argv[i]);
          break;

        default: printf("wrong argument near %s\n\n",argv[i]);
          exit(1);
        }
      } else {
        printf("wrong argument near %s\n\n",argv[i]);
        exit(1);
      }
  }

if (Strategy_nu!=2&&Strategy_nu!=4) {
  printf("only decoder strategies 2 and 4 allowed.\n\n");
  exit(1);
}

if (!(Mode&128||Mode&256||Mode&2048) && gap_GPC_wide==0) {
  printf("gap_GPC_wide=0 chosen in Mode other than EQF1,EQF2,EQF3, exciting..\n");
  exit(1);
}

if (distro && !encode_decode)
  printf("\n** warning: option distro (-v) only valid with encode_decode (-t)\n\n");

if (encode_decode && !Mode)
  {
    printf("\n** warning: Mode not chosen in encode_decode (-t)\n\n");
    exit(1);
  }
if (Mode && !encode_decode)
  printf("\n** warning: Mode chosen (-M) but not with encode_decode (-t), Mode not considered\n\n");


En = new Encode;
Rnd = new Random();
Gpc = new GPC;

print_parameters(argv);

print_mode(Mode);

starttime = print_current_time("start time is:",0);

if (logframes) init_log(&flog);

if ( optspread )
  /*
```

```
      get the spreading length for a burst of length sectors
      via the Encoder
   */
   {
     int frameend,framebeg=0;
     for(frame=0; frame<MaxFrame; frame++)
       {
         if ( frame < 200 || frame >= 200+length )
           setFramecontents(&En->inputFrame,FILLBYTE);
         else setFramecontents(&En->inputFrame,FILLBYTE-1);
         En->oneFrame();
         if ( frame > 115 && En->otherfill() )
           {
             INFO(4,("frame # %d contains symbols other than FILLBYTE!\n",frame));
             if (!framebeg) framebeg=frame;
             frameend=frame;
           }
       }
     printf("first occurence of modified frame: %d, last: %d, diff: %d\n",framebeg,frameend, frameend-framebeg
   }

 if ( optframe || optburst )
   {
     for(frame=0; frame<120; frame++)
       {
         setFramecontents(&En->inputFrame,FILLBYTE);
         En->oneFrame();
       }
     memcpy(orgFrame,En->outputFrame,FC1SZBCW*sizeof(int));
   }

 if (optframe)
   {
     INFO(1,("\ntest of frame patterns running...\n"));
     testFramepattern(Strategy_nu);
   }

 if (optburst)
   {
     INFO(1,("\nBurst testing of different patterns running...\n"));
     for (pnu=0;pnu<sizeof(pattern)/sizeof(int)/FC1SZBCW;pnu++)
       {
         INFO(4,("burst testing pattern # %d\n",pnu));
         testmaxBurst(pnu,Strategy_nu);
       }
   }

 if (encode_decode)
   {
     int j;
     int symbcntenc=0;
     int random_part = BER*BERFACTOR;
     struct Error_kinds errs={0,0,0,0,0,0,0,0,0,0};
     int encoded_copy[FC1SZB];

     De = new Decode(Strategy_nu);

     for (frame=0; frame<maxframes; frame++) {

       if(logframes)
         {
           fprintf(flog,"%d\n",frame);
           rewind(flog);
         }
       if (distro)
         {
           for (j=0;j<FSZB;j++)
             if (j>2&&j<5) En->inputFrame.byte[j] = frame;//symbcntenc++;
             else En->inputFrame.byte[j] = 0;
         }
       else
         for (j=0;j<FSZB;j++,symbcntenc++)
           En->inputFrame.byte[j] = (unsigned char) symbcntenc;
```

```
      //   setFramecontents(&En->inputFrame,FILLBYTE);

      En->oneFrame();

      if (distro) print_encoded_frame(De,En->outputFrame);

      if (distro) continue;

      memcpy(De->inputFrame,En->outputFrame,FC1SZBCW*sizeof(int));
      memcpy(encoded_copy,En->outputFrame,FC1SZBCW*sizeof(int));

      if (Mode&1) introduceRandomErrors(Rnd, De, random_part);
      if (Mode&2) introducePDFBursts(De->inputFrame, Rnd, thinning);
      if (Mode&4) introduceGPCErrors(De->inputFrame, Gpc, Rnd);
      if (Mode&8) introduceDoubleRandomErrors(Rnd, De, random_part);
      if (Mode&16) introduceTripleRandomErrors(Rnd, De, random_part);
      if (Mode&32) introduceEquidistantErrors(Rnd, De, gap_GPC_wide);
      if (Mode&64) introduceErrorSequence(Rnd,De,gap_GPC_wide,frame_offset);
      if (Mode&128) introduceFrameError(De->inputFrame,pattern[0],gap_GPC_wide);
      if (Mode&256) introduceFrameError(De->inputFrame,pattern[1],gap_GPC_wide);
      if (Mode&512) introduceErrorSequenceShifted(Rnd,De,gap_GPC_wide);
      if (Mode&1024) introduceErrorGroups(Rnd,De,gap_GPC_wide);
      if (Mode&2048) introduceFrameError(De->inputFrame,pattern[2],gap_GPC_wide);
      if (Mode&4096) introduceRandomErrorSymbols(Rnd,De,gap_GPC_wide);

      accumulate_overall_errors(encoded_copy,De->inputFrame);

      De->oneFrame();

      check_decoding_frame_result(De->outputFrame.byte,
                                  De->outerasFrame.byte,
                                  &errs);
    }

    print_current_time("stop time is:",starttime);
    [redacted]
          modified_symbols_PDF,not_modified_symbols_PDF,not_modified_symbols_PDF+modified_symbols_PDF,
          modified_symbols_PDF/(double)(not_modified_symbols_PDF+modified_symbols_PDF),
          intro_symbols, ok_symbols, intro_symbols+ok_symbols,
          intro_symbols/(double)(intro_symbols+ok_symbols),
          modified_symbols,not_modified_symbols,not_modified_symbols+modified_symbols,
          modified_symbols/(double)(not_modified_symbols+modified_symbols),
          overall_errors/(float)(overall_errors+overall_not_errors),
          errs.single_errors,
          errs.bytes_checked, errs.single_errors/(double)(errs.bytes_checked),
          errs.bytes_checked/FSZB,
          errs.interpolated_eras, errs.interpolated_eras/(double)(errs.bytes_checked),
          errs.error_clicks, errs.error_clicks/(double)(errs.bytes_checked),
          errs.errs_not_eras, errs.errs_not_eras/(double)(errs.bytes_checked),
          errs.P00, errs.P01, errs.P10, errs.P11,
          errs.P00/(double)(errs.bytes_checked), errs.P01/(double)(errs.bytes_checked),
          errs.P10/(double)(errs.bytes_checked), errs.P11/(double)(errs.bytes_checked));

    if (bursts)
      printf("  bursts %d\n  gaps %d\n=>average burst length: %f\n=>average gap length: %f\n\n",
             bursts, gaps,
             modified_symbols/(float)bursts, not_modified_symbols/(float)gaps);

    if(logframes) fclose(flog);
  }

if (encodeonly)
  {
    int j,symb=0,fd;
    if ((fd=open(filename,O_WRONLY|O_CREAT,S_IREAD|S_IWRITE)) < 0)
      {
        perror("encodeonly:open");
        exit(1);
      }
    for (frame=0; frame<MaxFrame; frame++) {
      for (j=0;j<FSZB;j++,symb++)
        En->inputFrame.byte[j] = (unsigned char) symb;
```

```
        En->oneFrame();
        write(fd,En->outputFrame,FC1SZBCW*sizeof(int));
      }
    }

  if(decodeonly)
    {
      struct Error_kinds errs={0,0,0,0,0,0,0,0,0};
      int fd;

      if ((fd=open(filename,O_RDONLY)) < 0)
        {
          perror("decodeonly:open");
          exit(1);
        }
      De = new Decode(Strategy_nu);
      for (frame=0; frame<MaxFrame; frame++)
        {
          if(logframes)
             {
              fprintf(flog,"%d\n",frame);
              rewind(flog);
             }

          read(fd,De->inputFrame,FC1SZBCW*sizeof(int));

          De->oneFrame();
          for(i=0;i<FSZB;i++)
            printf("%d ",De->outputFrame.byte[i]);
          check_decoding_frame_result(De->outputFrame.byte,
                                      De->outerasFrame.byte,
                                      &errs);
        }
      print_current_time("stop time is:",starttime);

      printf(" modified symbols(after encoding, PDF only): %d\n not modified symbols (PDF only): %d\n all sy
             modified_symbols_PDF,not_modified_symbols_PDF,not_modified_symbols_PDF+modified_symbols_PDF,
             modified_symbols_PDF/(double)(not_modified_symbols_PDF+modified_symbols_PDF),
             intro_symbols, ok_symbols, intro_symbols+ok_symbols,
             intro_symbols/(double)(intro_symbols+ok_symbols),
             errs.single_errors,
             errs.bytes_checked, errs.single_errors/(double)(errs.bytes_checked),
             errs.bytes_checked/FSZB,
             errs.interpolated_eras, errs.interpolated_eras/(double)(errs.bytes_checked),
             errs.error_clicks, errs.error_clicks/(double)(errs.bytes_checked),
             errs.errs_not_eras, errs.errs_not_eras/(double)(errs.bytes_checked),
             errs.P00, errs.P01, errs.P10, errs.P11,
             errs.P00/(double)(errs.bytes_checked), errs.P01/(double)(errs.bytes_checked),
             errs.P10/(double)(errs.bytes_checked), errs.P11/(double)(errs.bytes_checked));
      if (bursts)
        printf("  bursts %d\n  gaps %d\n=>average burst length: %f\n=>average gap length: %f\n\n",
               bursts, gaps,
               modified_symbols/(float)bursts, not_modified_symbols/(float)gaps);

      if(logframes) fclose(flog);
    }
}



/*
 * Encode.cc
 * Copyright Kay Rydyger
 *
 * Encoding class
 */

#include "defs.h"
#include "classes.h"

static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

Encode::Encode()
```

```
{
        int i;

        Efm = new EFM();
        RsEncoder  = new RSEncoder();

        Pos1=0; Pos2=2;
        Pos=0; PosBuf=0; Pos3=1;

        puffer  = (union C1Frame*) calloc(112,sizeof(union C1Frame));
        input   = (union C2Frame*) calloc(3,sizeof(union C2Frame));
        output  = (union C1Frame*) calloc(2,sizeof(union C1Frame));
        outputFrame  = (int*) calloc(FC1SZBCW,sizeof(int));


        memset(puffer,0,112*sizeof(union C1Frame));
        memset(input,0,3*sizeof(union C2Frame));
        memset(output,0,2*sizeof(union C1Frame));
        memset(&inputFrame,0,sizeof(inputFrame));
        memset(outputFrame,0,FC1SZBCW*sizeof(sizeof(int)));
        for (i=0;i<FSZB;i++) inputFrame.byte[i]=0;
}

Encode::~Encode()
{
        free(puffer);
        free(input);
        free(output);
        free(outputFrame);
        delete Efm;
        delete RsEncoder;
}


#ifndef SCRAMBLEOFF
void Encode::oneFrame()
{
        union C1Frame *c1;
        c1 = ScrambleFrame();
        EFMencodeFrame(c1);
}

#else
void Encode::oneFrame()
{
        int i;
        union C1Frame c1;
        union C2Frame c2;
        for (i=0;i<6;i++)
                c2.word[i] = inputFrame.word[i];
        for (i=6;i<FSZW;i++)
                c2.word[i+2] = inputFrame.word[i];
        RSENCODE(&c2);
        for (i=0;i<FC2SZW;i++)
                c1.word[i] = c2.word[i];
        RSENCODE(&c1);
        for (i=0; i<FC1SZB; i++)
                outputFrame[i] = TRAFOTOEFM(c1.byte[i]);
}
#endif

union C1Frame *Encode::ScrambleFrame()
{
        unsigned i,x,y;
        union C1Frame *c1frame;

        input[Pos2].word[0]  = inputFrame.word[0];
        input[Pos2].word[3]  = inputFrame.word[1];
        input[Pos1].word[8]  = inputFrame.word[2];
        input[Pos1].word[11] = inputFrame.word[3];
        input[Pos2].word[1]  = inputFrame.word[4];
        input[Pos2].word[4]  = inputFrame.word[5];
        input[Pos1].word[9]  = inputFrame.word[6];
```

```
        input[Pos1].word[12] = inputFrame.word[7];
        input[Pos2].word[2]  = inputFrame.word[8];
        input[Pos2].word[5]  = inputFrame.word[9];
        input[Pos1].word[10] = inputFrame.word[10];
        input[Pos1].word[13] = inputFrame.word[11];

        RSENCODE(&input[Pos1]);

        INC1R3(Pos1);
        INC1R3(Pos2);

        for ( x=0,y=PosBuf; x<28; x++) {
                puffer[y].byte[x] = input[Pos].byte[x];
                INC4R109(y);
        }

        c1frame = &puffer[PosBuf];

        RSENCODE(&puffer[PosBuf]);

        INC1R3(Pos);
        INC1R109(PosBuf);

        if (Pos3==0) for (i=0; i<FC1SZB; i++)
                output[i%2].byte[i] = c1frame->byte[i];
        else for (i=0; i<FC1SZB; i++)
                output[1-i%2].byte[i] = c1frame->byte[i];

        INC1R2(Pos3);

        return &output[Pos3];
}

void Encode::EFMencodeFrame(union C1Frame *c1)
{
        unsigned i;
        int *out = outputFrame;
        *out++ = TRAFOTOEFM(0);  // add one control word per frame
        for (i=0; i<FC1SZB; i++)
                *out++ = TRAFOTOEFM(c1->byte[i]);
}

void Encode::setChnbit(int i)
{
        if (i>560&&i<588) // this hides the sync pattern into the unused bits of frame.
        {                 // decoding is currently done by discarding them.
                outputFrame[i-561] |= (1<<17);
                return;
        }
        if (i<=560&&i>=0)
        {
                outputFrame[i/17] |= (1<<(i%17));
                return;
        }
        printf("Encode::setChnbit: wrong channel bit number, i=%d !!\n",i);
}

void Encode::toggleChnbit(int i)
{
        if (i>560&&i<588)
        {
                outputFrame[i-561] ^= (1<<17);
                return;
        }
        if (i<=560&&i>=0)
        {
                outputFrame[i/17] ^= (1<<(i%17));
                return;
        }
        printf("Encode::toggleChnbit: wrong channel bit number, i=%d !!\n",i);
}

int Encode::getChnbit(int i)
```

```
{
        if (i>560&&i<588)
                return (outputFrame[i-561]&(1<<17))?1:0;
        if (i<=560&&i>=0)
                return ((1<<(i%17))&outputFrame[i/17])?1:0;
        printf("Encode::getChnbit: wrong channel bit number, i=%d !!\n",i);
        return 0;
}


int Encode::otherfill()
{
        int i;

        for (i=1;i<13;i++)   //  number zero is CW !
                if ( TRAFOFROMEFM(outputFrame[i])!=FILLBYTE )
                        return 1;
        for (i=17;i<29;i++)
                if ( TRAFOFROMEFM(outputFrame[i])!=FILLBYTE )
                        return 1;
        return 0;
}

/*
 * Decode.c
 * Copyright Kay Rydyger
 *
 * Decode class
 *
 */

#include "defs.h"
#include "classes.h"


static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

union C2Frame Decode::eradumm;

Decode::Decode(int decoder_strat)
{
  Efm = new EFM();

  C1  = new C1Decoder(decoder_strat);
  C2  = new C2Decoder(decoder_strat);

  Pos=0;
  Pos1=1;
  Pos2=0;
  PosBuf=0;

  puffer  = (union C2Frame*) calloc(112,sizeof(union C2Frame));
  erapuf  = (union C2Frame*) calloc(112,sizeof(union C2Frame));
  input   = (union C1Frame*) calloc(2,sizeof(union C1Frame));
  inputera= (union C1Frame*) calloc(2,sizeof(union C1Frame));
  output  = (union Frame*) calloc(3,sizeof(union Frame));
  outputera  = (union Frame*) calloc(3,sizeof(union Frame));
  inputFrame = (int*) calloc(FC1SZBCW,sizeof(int));


  memset(puffer,0,112*sizeof(union C2Frame));
  memset(erapuf,0,112*sizeof(union C2Frame));
  memset(input,0,2*sizeof(union C1Frame));
  memset(inputera,0,2*sizeof(union C1Frame));
  memset(output,0,3*sizeof(union Frame));
  memset(outputera,0,3*sizeof(union Frame));
  memset(inputFrame,0,FC1SZBCW*sizeof(int));
  memset(&outputFrame,0,sizeof(outputFrame));
  memset(&eradumm,0,sizeof(eradumm));
}

Decode::~Decode()
{
```

```
        free(puffer);
        free(erapuf);
        free(input);
        free(inputera);
        free(output);
        free(inputFrame);
        fflush(stdout);

        delete Efm;
        delete C1;
        delete C2;
}


#ifndef SCRAMBLEOFF
void Decode::oneFrame()
{
        union C1Frame *c1;
        c1 = EFMdecodeFrame();
        DescrambleFrame(c1);
}

#else

void Decode::oneFrame()
{
  union C1Frame c1;
  union C2Frame c2;
  union C1Frame c1era;

  int i, ret;
  for (i=0;i<FC1SZB;i++)
    {
      ret = TRAFOFROMEFM(inputFrame[i+1]);   //  because number zero is CW !
      c1.byte[i] = (BYTE) ret;
      if ( ret == WRONGEFM ) c1era.byte[i] = 1;
      else c1era.byte[i] = 0;
    }

  C1DECODE(&c1,&c1era);
  for (i=0; i<FC2SZB; i++)
    {
      c2.byte[i] = c1.byte[i];
      eradumm.byte[i] = C1->geterasure(i);
    }

  C2DECODE(&c2,&eradumm);
  for (i=0; i<6; i++)
    outputFrame.word[i] = c2.word[i];
  for (i=8; i<FC2SZW; i++)
    outputFrame.word[i-2] = c2.word[i];
}

#endif

union C1Frame *Decode::EFMdecodeFrame()
{
  unsigned int i, oldPos, ret;
  if (Pos==0)
    for (i=0; i<FC1SZB; i++)
      {
        ret = TRAFOFROMEFM(inputFrame[i+1]);  //  because number zero is CW !
        input[i%2].byte[i] = ret;
        if ( ret == WRONGEFM ) inputera[i%2].byte[i] = 1;
        else inputera[i%2].byte[i] = 0;
      }
  else
    for (i=0; i<FC1SZB; i++)
      {
        ret = TRAFOFROMEFM(inputFrame[i+1]);
        input[1-(i%2)].byte[i] = ret;
        if ( ret == WRONGEFM ) inputera[1-(i%2)].byte[i] = 1;
        else inputera[1-(i%2)].byte[i] = 0;
```

```
    }

  INFO(2,("\n\n\n============    FRAME  NR.: %d    ================\n",frame));
  SHOWPUFFERC1(1<<9,"Decode: input-puffer\n",input);
  SHOWPUFFERC1(1<<9,"Decode: erasure-puffer\n",inputera);

  INFO(2,("Decode: C1 decoding line %d in input-puffer\n\n",Pos));

  C1DECODE(&input[Pos],&inputera[Pos]);

  oldPos = Pos;
  INC1R2(Pos);

  return &input[oldPos];
}

void Decode::DescrambleFrame(union C1Frame *input)
{
        union C2Frame *c2frame, *c2eras;
        int x,y,i,j;

        for (x=0,y=PosBuf;x<FC2SZB; x++) {

                puffer[y].byte[x] = input->byte[x];
                erapuf[y].byte[x] = C1->geterasure(x);

                DEC4R109(y);
        }

        SHOWPUFFERC2(1<<11,"Decode: erapuf\n",erapuf);
        SHOWPUFFERC2(1<<9,"Decode: puffer\n",puffer);

        INC4R109(y);

        INFO(2,("Decode: C2 decoding line %d in puffer\n",y));

        C2DECODE(&puffer[y],&erapuf[y]);

        SHOWPUFFERC2(1<<11,"Decode: after decode: erapuf\n",erapuf);
        SHOWPUFFERC2(1<<9,"Decode: after decode: puffer\n",puffer);

        INC1R109(PosBuf);

        c2frame = &puffer[y];
        c2eras = &erapuf[y];

        output[Pos1].word[0]  =  c2frame->word[0];
        output[Pos1].word[4]  =  c2frame->word[1];
        output[Pos1].word[8]  =  c2frame->word[2];
        output[Pos1].word[1]  =  c2frame->word[3];
        output[Pos1].word[5]  =  c2frame->word[4];
        output[Pos1].word[9]  =  c2frame->word[5];
        output[Pos2].word[2]  =  c2frame->word[8];
        output[Pos2].word[6]  =  c2frame->word[9];
        output[Pos2].word[10] =  c2frame->word[10];
        output[Pos2].word[3]  =  c2frame->word[11];
        output[Pos2].word[7]  =  c2frame->word[12];
        output[Pos2].word[11] =  c2frame->word[13];

        // scramble erasures as well
        outputera[Pos1].word[0]  =  c2eras->word[0];
        outputera[Pos1].word[4]  =  c2eras->word[1];
        outputera[Pos1].word[8]  =  c2eras->word[2];
        outputera[Pos1].word[1]  =  c2eras->word[3];
        outputera[Pos1].word[5]  =  c2eras->word[4];
        outputera[Pos1].word[9]  =  c2eras->word[5];
        outputera[Pos2].word[2]  =  c2eras->word[8];
        outputera[Pos2].word[6]  =  c2eras->word[9];
        outputera[Pos2].word[10] =  c2eras->word[10];
        outputera[Pos2].word[3]  =  c2eras->word[11];
        outputera[Pos2].word[7]  =  c2eras->word[12];
        outputera[Pos2].word[11] =  c2eras->word[13];
```

```
        SHOWPUFFER_FC2SZB(1<<9,"Decode: c2frame-puffer\n",c2frame);
        memcpy(&outputFrame,&output[Pos1],FSZB);
        memcpy(&outerasFrame,&outputera[Pos1],FSZB);
        SHOWPUFFER_FSZB(1<<9,"Decode: output-puffer\n",output);

        SHOWPUFFER_FC2SZB(1<<11,"Decode: c2eras-puffer\n",c2eras);
        SHOWPUFFER_FSZB(1<<11,"Decode: outputera-puffer\n",outputera);

        INC1R3(Pos1);
        INC1R3(Pos2);
}

void Decode::setChnbit(int i)
{
        if (i>560&&i<588)
        {
                inputFrame[i-561] |= (1<<17);
                return;
        }
        if (i<=560&&i>=0)
        {
                inputFrame[i/17] |= (1<<(i%17));
                return;
        }
        printf("Decode::setChnbit: wrong channel bit number, i=%d !!\n",i);
}

void Decode::toggleChnbit(int i)
{
        if (i>560&&i<588)
        {
                inputFrame[i-561] ^= (1<<17);
                return;
        }
        if (i<=560&&i>=0)
        {
                inputFrame[i/17] ^= (1<<(i%17));
                return;
        }
        printf("Decode::toggleChnbit: wrong channel bit number, i=%d !!\n",i);
}

int Decode::getChnbit(int i)
{
        if (i>560&&i<588)
                return (inputFrame[i-561]&(1<<17))?1:0;
        if (i<=560&&i>=0)
                return ((1<<(i%17))&inputFrame[i/17])?1:0;
        printf("Decode::getChnbit: wrong channel bit number, i=%d !!\n",i);
        return 0;
}


int Decode::modified(void)
{
        int i;
        for (i=0;i<FSZB;i++)
                if (outputFrame.byte[i]!=FILLBYTE) return 1;
        return 0;
}


/*
 * EFM.c
 * Copyright Kay Rydyger
 *
 * EFM modulation and demodulation
 *
 */

#include "defs.h"
#include "classes.h"
```

```
#include "EFMTable.h"

static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

static int useofficialEFMTable=1;

int maxNonEFMWords;

EFM::EFM()
{
  if (useofficialEFMTable)
    CreateOfficialEFMTable();
  else
    CreateMyEFMTable();
  maxNonEFMWords = CreateNonEFMWords();
  DSV = 0;
}

int EFM::TrafoToEFM(BYTE word)
{
        unsigned int efmword;
        efmword = EFMTable[word];
        (void) add_merging_bits(&efmword);
        return TO_NRZI(efmword);
}

int EFM::TrafoFromEFM(int word)
{
        unsigned int ret, fromnrzi;
        fromnrzi = FROM_NRZI(word);
        ret = EFMrevTable[fromnrzi&mask14];
        if ( ret==mask14 )
        {
                INFO(1<<4,("TrafoFromEFM: wrong EFM code ! EFMrevTable[%d]=%d, word=%d\n",
                        fromnrzi&mask14,ret,word));
                return WRONGEFM; // for setting erasures !
        }
        return  ret;
}

void EFM::printEFMTable()
{
        int i, byte;

        for(i=0; i<256; i++)
        {
                byte = EFMTable[i];
                printf("EFMtable[ %d ] = %d%d%d%d%d%d%d%d%d%d%d%d%d %d\n",i,
                        (byte>>13)&1,(byte>>12)&1,(byte>>11)&1,(byte>>10)&1,(byte>>9)&1,
                        (byte>>8)&1,(byte>>7)&1,(byte>>6)&1,(byte>>5)&1,(byte>>4)&1,(byte>>3)&1,
                        (byte>>2)&1,(byte>>1)&1,(byte>>0)&1, byte);
       }
        for(i=0; i<=9400; i++)  // !!! max value is 9362 for EFMTable !!!
        {
                byte = EFMrevTable[i];
                if ( byte != mask14 )
                        printf("EFMrevTable[ %d %d%d%d%d%d%d%d%d%d%d%d%d%d ] =  %d\n",i,
                                (i>>13)&1,(i>>12)&1,(i>>11)&1,(i>>10)&1,(i>>9)&1,
                                (i>>8)&1,(i>>7)&1,(i>>6)&1,(i>>5)&1,(i>>4)&1,(i>>3)&1,
                                (i>>2)&1,(i>>1)&1,(i>>0)&1, byte);

        }
        printf("************************\n");
        for (i=0;i<256;i++)
          {
            byte = EFMTable[i];
                printf("%d\t%d%d%d%d%d%d%d%d%d%d%d%d%d\n",i,
                        (byte>>13)&1,(byte>>12)&1,(byte>>11)&1,(byte>>10)&1,(byte>>9)&1,
                        (byte>>8)&1,(byte>>7)&1,(byte>>6)&1,(byte>>5)&1,(byte>>4)&1,(byte>>3)&1,
                        (byte>>2)&1,(byte>>1)&1,(byte>>0)&1);

        }
```

```
        printf("------------------\n");
        for(i=0; i<=9400; i++)
        {
                byte = EFMrevTable[i];
                if ( byte != mask14 )
                        printf("%d\t%d%d%d%d%d%d%d%d%d%d%d%d%d%d\n",byte,
                                (i>>13)&1,(i>>12)&1,(i>>11)&1,(i>>10)&1,(i>>9)&1,
                                (i>>8)&1,(i>>7)&1,(i>>6)&1,(i>>5)&1,(i>>4)&1,(i>>3)&1,
                                (i>>2)&1,(i>>1)&1,(i>>0)&1);

        }

        fflush(stdout);
}

int EFM::getNonEFMWords(int i)
{
  return NonEFMWordsTable[i];
}

/* ------------ protected member functions --------- */

int EFM::MB[4] = { 0,1<<14,1<<15,1<<16 };

void EFM::CreateOfficialEFMTable()
{
  int i,nr,j;
  char efm_input_str[16];
  for (i=0; i<maxft14; i++) EFMrevTable[i] = mask14;
  for (i=0;i<256;i++)
    {
      sscanf(EFMTable_inc[i],"%d %s",&nr,efm_input_str);
      EFMTable[nr]=0;
      for(j=0;j<14;j++)
        EFMTable[nr]|=(efm_input_str[13-j]=='1')?1<<j:0;
      EFMrevTable[EFMTable[nr]] = nr;
    }
}

int EFM::CreateNonEFMWords(void)
{
  int i,j;
  for (i=0,j=0;i<maxft14; i++)
    {
      if (EFMrevTable[i]==mask14)
        NonEFMWordsTable[j++]=to_NRZI(i);
    }
  return j;
}

int EFM::test_constraints(int code)
{
        int i, bit;
        int nullen=0, einsen=0, start=1;

        for(i=0;i<ft;i++)
        {
                bit = ((code>>i)&1);
                if (!bit)
                {
                        nullen++;
                        einsen=0;
                }
                if ( nullen >= kkk ) return 0;  /* k-constraint */
                if (bit)
                {
                        if ( nullen < ddd-1 && nullen > 0 && !start) return 0;
/*
   d-constraint */
                        start=0;
                        nullen=0;
                        einsen++;
                        if ( einsen > 1 ) return 0;
```

```
                }
        }
        return 1;
}

void EFM::CreateMyEFMTable(void)
{
        int i, code, z=0;

        for (i=0; i<maxft14; i++) EFMrevTable[i] = mask14;
        for (code=0; code<maxft14; code++)
        {
                if ( !test_constraints(code) ) continue;
                if ( calc_aNulls(code) >= 8 || code == 9362 || code == 9361 ) continue;
                    /* these patterns (first term) are excluded because of possible
                       generation of sync patterns when connected to the right successor
                    */
                EFMTable[z] = code;
                EFMrevTable[code] = z;
                z++;
        }
//        return z;
}

int EFM::to_NRZI(unsigned int EfmMb)
{
        static int updown = 0;
        int i, nrzi=0;

        for (i=svteen-1;i>=0;i--)
        {
                if ( ((EfmMb>>i)&1) == 1 ) updown = 1 - updown;
                if (updown) my_setbit(nrzi,i);
        }
        return nrzi;
}

int EFM::from_NRZI(int nrzi)
{
        static int old = 0 ;
        int i, nw, efmmb=0;

        for (i=svteen-1; i>=0; i-- )
        {
                nw = (nrzi>>i)&1;
                if ( nw != old ) my_setbit(efmmb,i);
                old = nw;
        }
        return efmmb;
}

int EFM::calcDSV(unsigned int efmword, int *d)
{
        int i, dsv=0, dflag = UP;
        for(i=ft-1;i>=0;i--)
        {
                if ( ((efmword>>i)&1) == 0 ) dsv += dflag;
                else dflag = -dflag;
        }
        *d = dflag;
        return dsv;
}


int EFM::calc_aNulls(unsigned efmword)
{
        int i;
        for(i=0;i<ft;i++) if ( (efmword>>i)&1 ) break;
        return i;
}

int EFM::calc_bNulls(unsigned efmword)
{
```

```
        int i;
        for(i=ft-1;i>=0;i--) if ( my_getbit(efmword,i) ) break;
        return ft-i-1;
}

int EFM::add_merging_bits(unsigned int *efmword)
{
        static unsigned int last_word = 9360; /* i.e. 10010010010000 */
        static int dflag = UP;
        int predDSV = LONG_MAX, newDSV, dflagtmp = dflag;
        int anulls, bnulls, MBcomb=9999, predDSVtmp;

        INFO(1<<3,("\nadd MB: word %d, last %d\n",*efmword, last_word));

        anulls = calc_aNulls(last_word);
        bnulls = calc_bNulls(*efmword);

        newDSV = calcDSV(*efmword,&dflagtmp);

        INFO(1<<3,("anulls %d bnulls %d newDSV %d dflagtmp %d\n",anulls,bnulls,newDSV,dflagtmp));

        if ( anulls <= 7 && bnulls <= 7-anulls )
        {
                /* !!!  AES p.121: EFMtable[ 119 ] = 01000000000010 4098 + 000:
                   might generate sync pattern !!!
                */

                predDSV = DSV + dflag * (3 + newDSV);
                MBcomb = 0;
                INFO(1<<3,(" in MBcomb=0, predDSV %d\n",predDSV));

        }

        if ( anulls <= 8 && bnulls <= 10 && bnulls > 1 )
        {
                /* second term (anulls != 8 || bnulls != 10) is to avoid sync
                   patterns, hope this is all ! look for sync patt between words ? !
                   last term for minimum constraint length
                */
                predDSVtmp = DSV + dflag * (2 - newDSV);

                INFO(1<<3,(" in MBcomb=1, predDSVtmp %d\n",predDSVtmp));
                if ( (abs(predDSVtmp) <= abs(predDSV)) || NOSUPPR )
                {
                        /* AES,p.250: must be "<=" and this order,
                           because MB with a transition are preferred.
                        */

                        predDSV = predDSVtmp;
                        MBcomb = 1;
                        dflag = -dflag;
                        INFO(1<<3,("set MBcomb=1!\n"));

                }
        }

        if ( anulls <= 9 && bnulls <= 9 && anulls > 0 && bnulls > 0 )
        {
                predDSVtmp = DSV + dflag * (0 - newDSV);

                INFO(1<<3,(" in MBcomb=2, predDSVtmp %d\n",predDSVtmp));
                if ( (abs(predDSVtmp) <= abs(predDSV)) || NOSUPPR )
                {
                        predDSV = predDSVtmp;
                        MBcomb = 2;
                        dflag = -dflag;
                        INFO(1<<3,("set MBcomb=2!\n"));
                }
        }

        if ( anulls <= 10 && bnulls <= 8 && anulls > 1 )
        {
                predDSVtmp = DSV + dflag * (0 - newDSV - 2);
```

```
                INFO(1<<3,(" in MBcomb=3, predDSVtmp %d\n",predDSVtmp));
                if ( (abs(predDSVtmp) <= abs(predDSV)) || NOSUPPR )
                {
                        predDSV = predDSVtmp;
                        MBcomb = 3;
                        dflag = -dflag;
                        INFO(1<<3,("set MBcomb=3!\n"));
                }
        }
        if (MBcomb==9999)
        {
                printf("EFM::add_merging_bits: internal error: *** no decision taken ***\n");
                printf("anulls=%d, bnulls=%d\n",anulls,bnulls);
                exit(1);
        }
        last_word = *efmword;
        dflag *= dflagtmp;
        DSV = predDSV;
        *efmword |= MB[MBcomb];
        return *efmword;
}

/*
 * C1.c
 * Copyright Kay Rydyger
 *
 * C1 decoder
 *
 */

#include "defs.h"
#include "classes.h"

static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

C1Decoder::C1Decoder(int decoder_strat)
{
        int i;
        for (i=0; i<nn; i++) {
                r[i] = ZERO;
                eraflags[i] = 0;
                sigmaji[i][0] = 0;
        }
        if (decoder_strat==2) do_decode=&C1Decoder::do_decode_strat2;
        if (decoder_strat==4) do_decode=&C1Decoder::do_decode_strat4;
}

void C1Decoder::C1Decode(union C1Frame *c1, union C1Frame *c1era)
{
  int i;
  for (i=0; i<FC1SZB-(nn-kk); i++)
    {
      r[i+nn-kk] = c1->byte[i];
      eraflags[i+nn-kk] = c1era->byte[i];
    }
  for (i=FC1SZB-(nn-kk); i<FC1SZB; i++)
    {
      r[i-FC1SZB+nn-kk] = c1->byte[i];
      eraflags[i-FC1SZB+nn-kk] = c1era->byte[i];
    }

  INFO(2,("C1 decoding Frame #:%d\n",frame));
  PSHOW(1<<7,"C1: erasures",eraflags);
  PSHOW(1<<7,"decode C1:",r);

  (this->*do_decode)();
  PSHOW(1<<7,"..decode C1 corr:",vcorr);
  for (i=0; i<FC1SZB-(nn-kk); i++)
    c1->byte[i] = vcorr[i+nn-kk];
}

// ------------  protected member functions -----------
```

```
void C1Decoder::do_decode_strat2() {

  GalEl E[nn+1];       // error correcting polynomial in frequency domain
  GalEl L[nn];         // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu;

  ret = get_syndromes1_2t(r,E);     // zero error syndrome
  if ( ret == 0 )
    {
      INFO(2,("C1:zero error syndrome \n"));
      memcpy(vcorr,r,nn);
      setoutputeraflags(0);
      return;
    }
  berlekamp(E,L,noeraflags);
  PSHOW(1<<6,"C1:error locator polynomial L:",L);
  nu = chien_search(L,X);

  if ( nu == 1 ) // single error syndrome
    {
      INFO(2,("C1:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      setoutputeraflags(0);
      return;
    }
  INFO(2,("C1:multiple  error syndrome nu=%d, assigning erasures..\n",nu));
  memcpy(vcorr,r,nn);
  setoutputeraflags(1);  // war: correct_with_fourney(E,L,r);
}



// Superstategy

void C1Decoder::do_decode_strat5() {

  GalEl E[nn+1];       // error correcting polynomial in frequency domain
  GalEl L[nn];         // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu;

  ret = get_syndromes1_2t(r,E);     // zero error syndrome
  if ( ret == 0 )
    {
      INFO(2,("C1:zero error syndrome \n"));
      memcpy(vcorr,r,nn);
      setoutputeraflags(0);
      return;
    }
  berlekamp(E,L,noeraflags);
  PSHOW(1<<6,"C1:error locator polynomial L:",L);
  nu = chien_search(L,X);

  //        PSHOW("C1:locator polynomial loc:",loc);
  //        for(i=0,nu=1;i<nn;i++) if ( loc[i] == ZERO ) X[nu++] = i;
  //        nu--;

  if ( nu == 1 ) // single error syndrome
    {
      INFO(2,("C1:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      setoutputeraflags(0);
      return;
    }
  if ( nu == 2 )  // double error syndrome
    {
      INFO(2,("C1:double error syndrome nu=2, trying to decode and assigning erasures..\n"));
```

```
      berlekamp(E,L,eraflags);
      nu = chien_search(L,X);
      //                PSHOW("C1:locator polynomial loc:",loc);
      //                for(i=0,nu=1;i<nn;i++) if ( loc[i] == ZERO ) X[nu++] = i;
      //                nu--;
      correct_with_fourney(E,L,X,nu);
      setoutputeraflags(1);
      return;
    }
  INFO(2,("C1:multiple (more than two)  error syndrome nu=%d, assigning erasures..\n",nu));
  memcpy(vcorr,r,nn);
  setoutputeraflags(1);
}


// ******************************STARTEGY 4*******************************

void C1Decoder::do_decode_strat4() {

  GalEl E[nn+1];       // error correcting polynomial in frequency domain
  GalEl L[nn];         // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu, nuoferas;

  ret = get_syndromes1_2t(r,E);     // zero error syndrome
  if ( ret == 0 )
    {
      INFO(2,("C1:zero error syndrome \n"));
      memcpy(vcorr,r,nn);
      setoutputeraflags(0);
      return;
    }
  berlekamp(E,L,noeraflags);
  PSHOW(1<<6,"C1:error locator polynomial L:",L);
  nu = chien_search(L,X);

  if ( nu == 1 ) // single error syndrome
    {
      INFO(2,("C1:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[i][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      setoutputeraflags(0);
      return;
    }
  nuoferas=0;
  for (i=0;i<nn;i++) if (eraflags[i]) nuoferas++;

  if ( nuoferas > 2 )
    {
      // copy output C1 erasure flags from input C1 erasure flags
      // done: C1::geterasure uses eraflags[] again.
      //      printf("C1:number of input erasures > 2 (nuoferas=%d)\n",nuoferas);
      INFO(2,("C1:number of input erasures > 2 (nuoferas=%d)\n",nuoferas));
      memcpy(vcorr,r,nn);
      return;
    }
  if ( nuoferas == 2 ) //&& nu == 1 )   ??
    {

      INFO(2,("C1:number of input erasures == 2, trying two erasure decoding\n",nuoferas));
      ret = berlekamp(E,L,eraflags);
      nu = chien_search(L,X);
      //                PSHOW("C1:locator polynomial loc:",loc);
      //                for(i=0,nu=1;i<nn;i++) if ( loc[i] == ZERO ) X[nu++] = i;
      //                nu--;
      if ( !ret && !correct_with_fourney(E,L,X,nu) )
        {
          INFO(2,("C1: nuoferas=2 and berlekamp and fourney correct ! erasures deleted\n"));
          setoutputeraflags(0);
          return;
        }
    }
```

```
  INFO(2,("C1: number of input erasures < 2 (nuoferas=%d) || (nuoferas==2 && (fourney or berlkamp failed)), ass
  memcpy(vcorr,r,nn);
  setoutputeraflags(1);
}

//*************************strategy 4 END*******************************


void C1Decoder::do_decode_stratmy() {

  GalEl E[nn+1];        // error correcting polynomial in frequency domain
  GalEl L[nn];          // error locating polynomial
  GalEl Y, X[nn];
  int ret, nu;

  ret = get_syndromes1_2t(r,E);     // zero error syndrome
  if ( ret == 0 )
    {
      INFO(2,("C1:zero error syndrome \n"));
      memcpy(vcorr,r,nn);
      setoutputeraflags(0);
      return;
    }
  berlekamp(E,L,noeraflags);
  PSHOW(1<<6,"C1:error locator polynomial L:",L);
  nu = chien_search(L,X);
  PSHOW(1<<6,"C1: X",X);

  if ( nu == 1 ) // single error syndrome
    {
      INFO(2,("C1:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      /*
        if ( get_syndromes1_2t(vcorr,E) )
        {

        INFO(2,("C1: wrong correction, assigning erasures and discard correction\n"));
        setoutputeraflags(1);
        memcpy(vcorr,r,nn);
        return;
        }
 */
  setoutputeraflags(0);
  return;
    }
        if ( nu == 2 )  // double error syndrome
          {
                INFO(2,("C1:double error syndrome nu=2, trying to decode and assigning erasures..\n"));
                ret = berlekamp(E,L,eraflags);
                nu = chien_search(L,X);

                if ( ret || correct_with_fourney(E,L,X,nu) )
                {
                        INFO(2,("C1: according to berlekamp or fourney: word is undecodable, no correction made
                        memcpy(vcorr,r,nn);
                }
                setoutputeraflags(1);
                return;
        }
        INFO(2,("C1:multiple (more than two)  error syndrome nu=%d, assigning erasures..\n",nu));
        memcpy(vcorr,r,nn);
        setoutputeraflags(1);
}


int C1Decoder::chien_search(GalEl *FTin, GalEl *X)
{
        GalEl  sum;
        int i,k,nu;
```

```
        sum = FTin[nn-1];
        nu = 1;
        for( k=nn-2; k>=0; k-- ) sum = Gadd(sum,FTin[k]);
        if ( sum == ZERO ) X[nu++] = 0;
            //out[0] = sum; // since inv(a0) = a0
        for( i=1; i<nn; i++ )
        {
                sum = FTin[nn-1];
                for( k=nn-2; k>=0; k-- ) sum = Gadd(Gmul(sum,ZERO-i),FTin[k]);
                if ( sum == ZERO ) X[nu++] = i;
                    //out[i] = sum;
        }
        nu--;
        return nu;
}

int C1Decoder::correct_with_fourney(GalEl *E, GalEl *L, GalEl *X, int nu)
{
            /* according to Peterson, Weldon: "Error correcting codes" p.297
             */
        int i,j,l;
        GalEl sum1, sum2;
        GalEl Y;

        if ( nu >= dd )
        {
                // word is undecodable !
        *       return 1;
        }

        PSHOW(1<<6,"C1: X",X);
        for (j=1; j<=nu; j++)
                for (i=1; i<nu; i++)
                        sigmaji[j][i] = Gadd(L[i],Gmul(X[j],sigmaji[j][i-1]));

        memcpy(vcorr,r,nn);
        for (j=1; j<=nu; j++)
        {
                sum1 = sum2 = ZERO;
                for (l=0; l<nu; l++) sum1 = Gadd(sum1,Gmul(sigmaji[j][l],E[nu-l]));
                for (l=0; l<nu; l++)
                        sum2 = Gadd(sum2,Gmul(sigmaji[j][l],Gpow(X[j],nu-l)));
                Y = Gdiv(sum1, sum2);
                vcorr[X[j]] = Gadd(r[X[j]], Y);
        }
        return 0;
}

void C1Decoder::setoutputeraflags(int i)
{
  memset(eraflags,i,FC2SZB);
}

int C1Decoder::geterasure(int i)
{
  return eraflags[i+nn-kk];
}

/*
 * C1.c
 * Copyright Kay Rydyger
 *
 * C1 decoder
 *
 */

#include "defs.h"
#include "classes.h"

static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

C1Decoder::C1Decoder(int decoder_strat)
{
```

```
  int i;
  for (i=0; i<nn; i++) {
    r[i] = ZERO;
    eraflags[i] = 0;
    sigmaji[i][0] = 0;
  }
  if (decoder_strat==2) do_decode=&C1Decoder::do_decode_strat2;
  if (decoder_strat==4) do_decode=&C1Decoder::do_decode_strat4;
}

void C1Decoder::C1Decode(union C1Frame *c1, union C1Frame *c1era)
{
  int i;
  for (i=0; i<FC1SZB-(nn-kk); i++)
    {
      r[i+nn-kk] = c1->byte[i];
      eraflags[i+nn-kk] = c1era->byte[i];
    }
  for (i=FC1SZB-(nn-kk); i<FC1SZB; i++)
    {
      r[i-FC1SZB+nn-kk] = c1->byte[i];
      eraflags[i-FC1SZB+nn-kk] = c1era->byte[i];
    }

  INFO(2,("C1 decoding Frame #:%d\n",frame));
  PSHOW(1<<7,"C1: erasures",eraflags);
  PSHOW(1<<7,"decode C1:",r);

  (this->*do_decode)();
  PSHOW(1<<7,"..decode C1 corr:",vcorr);
  for (i=0; i<FC1SZB-(nn-kk); i++)
    c1->byte[i] = vcorr[i+nn-kk];
}

// ------------  protected member functions -----------

void C1Decoder::do_decode_strat2() {

  GalEl E[nn+1];       // error correcting polynomial in frequency domain
  GalEl L[nn];         // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu;

  ret = get_syndromes1_2t(r,E);    // zero error syndrome
  if ( ret == 0 )
    {
      INFO(2,("C1:zero error syndrome \n"));
      memcpy(vcorr,r,nn);
      setoutputeraflags(0);
      return;
    }
  berlekamp(E,L,noeraflags);
  PSHOW(1<<6,"C1:error locator polynomial L:",L);
  nu = chien_search(L,X);

  if ( nu == 1 ) // single error syndrome
    {
      INFO(2,("C1:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      setoutputeraflags(0);
      return;
    }
  INFO(2,("C1:multiple  error syndrome nu=%d, assigning erasures..\n",nu));
  memcpy(vcorr,r,nn);
  setoutputeraflags(1);  // war: correct_with_fourney(E,L,r);
}



// Superstategy
```

```
void C1Decoder::do_decode_strat5() {

  GalEl E[nn+1];       // error correcting polynomial in frequency domain
  GalEl L[nn];         // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu;

  ret = get_syndromes1_2t(r,E);    // zero error syndrome
  if ( ret == 0 )
    {
      INFO(2,("C1:zero error syndrome \n"));
      memcpy(vcorr,r,nn);
      setoutputeraflags(0);
      return;
    }
  berlekamp(E,L,noeraflags);
  PSHOW(1<<6,"C1:error locator polynomial L:",L);
  nu = chien_search(L,X);

  //        PSHOW("C1:locator polynomial loc:",loc);
  //        for(i=0,nu=1;i<nn;i++) if ( loc[i] == ZERO ) X[nu++] = i;
  //        nu--;

  if ( nu == 1 ) // single error syndrome
    {
      INFO(2,("C1:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      setoutputeraflags(0);
      return;
    }
  if ( nu == 2 )  // double error syndrome
    {
      INFO(2,("C1:double error syndrome nu=2, trying to decode and assigning erasures..\n"));
      berlekamp(E,L,eraflags);
      nu = chien_search(L,X);
      //                PSHOW("C1:locator polynomial loc:",loc);
      //                for(i=0,nu=1;i<nn;i++) if ( loc[i] == ZERO ) X[nu++] = i;
      //                nu--;
      correct_with_fourney(E,L,X,nu);
      setoutputeraflags(1);
      return;
    }
  INFO(2,("C1:multiple (more than two)  error syndrome nu=%d, assigning erasures..\n",nu));
  memcpy(vcorr,r,nn);
  setoutputeraflags(1);
}


// ********************************STARTEGY 4*********************************

void C1Decoder::do_decode_strat4() {

  GalEl E[nn+1];       // error correcting polynomial in frequency domain
  GalEl L[nn];         // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu, nuoferas;

  ret = get_syndromes1_2t(r,E);    // zero error syndrome
  if ( ret == 0 )
    {
      INFO(2,("C1:zero error syndrome \n"));
      memcpy(vcorr,r,nn);
      setoutputeraflags(0);
      return;
    }
  berlekamp(E,L,noeraflags);
  PSHOW(1<<6,"C1:error locator polynomial L:",L);
  nu = chien_search(L,X);

  if ( nu == 1 ) // single error syndrome
    {
```

```
      INFO(2,("C1:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      setoutputeraflags(0);
      return;
    }
  nuoferas=0;
  for (i=0;i<nn;i++) if (eraflags[i]) nuoferas++;

  if ( nuoferas > 2 )
    {
      // copy output C1 erasure flags from input C1 erasure flags
      // done: C1::geterasure uses eraflags[] again.
      //      printf("C1:number of input erasures > 2 (nuoferas=%d)\n",nuoferas);
      INFO(2,("C1:number of input erasures > 2 (nuoferas=%d)\n",nuoferas));
      memcpy(vcorr,r,nn);
      return;
    }
  if ( nuoferas == 2 ) //&& nu == 1 )   ??
    {

      INFO(2,("C1:number of input erasures == 2, trying two erasure decoding\n",nuoferas));
      ret = berlekamp(E,L,eraflags);
      nu = chien_search(L,X);
      //                PSHOW("C1:locator polynomial loc:",loc);
      //                for(i=0,nu=1;i<nn;i++) if ( loc[i] == ZERO ) X[nu++] = i;
      //                nu--;
      if ( !ret && !correct_with_fourney(E,L,X,nu) )
        {
          INFO(2,("C1: nuoferas=2 and berlekamp and fourney correct ! erasures deleted\n"));
          setoutputeraflags(0);
          return;
        }
    }

  INFO(2,("C1: number of input erasures < 2 (nuoferas=%d) || (nuoferas==2 && (fourney or berlkamp failed)), ass
  memcpy(vcorr,r,nn);
  setoutputeraflags(1);
}

//*************************strategy 4 END*******************************


void C1Decoder::do_decode_stratmy() {

  GalEl E[nn+1];        // error correcting polynomial in frequency domain
  GalEl L[nn];          // error locating polynomial
  GalEl Y, X[nn];
  int ret, nu;

  ret = get_syndromes1_2t(r,E);     // zero error syndrome
  if ( ret == 0 )
    {
      INFO(2,("C1:zero error syndrome \n"));
      memcpy(vcorr,r,nn);
      setoutputeraflags(0);
      return;
    }
  berlekamp(E,L,noeraflags);
  PSHOW(1<<6,"C1:error locator polynomial L:",L);
  nu = chien_search(L,X);
  PSHOW(1<<6,"C1: X",X);

  if ( nu == 1 ) // single error syndrome
    {
      INFO(2,("C1:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      setoutputeraflags(0);
      return;
    }
```

```
  if ( nu == 2 )  // double error syndrome
    {
      INFO(2,("C1:double error syndrome nu=2, trying to decode and assigning erasures..\n"));
      ret = berlekamp(E,L,eraflags);
      nu = chien_search(L,X);

      if ( ret || correct_with_fourney(E,L,X,nu) )
        {
          INFO(2,("C1: according to berlekamp or fourney: word is undecodable, no correction made and assigning
          memcpy(vcorr,r,nn);
        }
      setoutputeraflags(1);
      return;
    }
  INFO(2,("C1:multiple (more than two)  error syndrome nu=%d, assigning erasures..\n",nu));
  memcpy(vcorr,r,nn);
  setoutputeraflags(1);
}


int C1Decoder::chien_search(GalEl *FTin, GalEl *X)
{
  GalEl  sum;
  int i,k,nu;

  sum = FTin[nn-1];
  nu = 1;
  for( k=nn-2; k>=0; k-- ) sum = Gadd(sum,FTin[k]);
  if ( sum == ZERO ) X[nu++] = 0;
  //out[0] = sum; // since inv(a0) = a0
  for( i=1; i<nn; i++ )
    {
      sum = FTin[nn-1];
      for( k=nn-2; k>=0; k-- ) sum = Gadd(Gmul(sum,ZERO-i),FTin[k]);
      if ( sum == ZERO ) X[nu++] = i;
      //out[i] = sum;
    }
  nu--;
  return nu;
}

int C1Decoder::correct_with_fourney(GalEl *E, GalEl *L, GalEl *X, int nu)
{
  /* according to Peterson, Weldon: "Error correcting codes" p.297
   */
  int i,j,l;
  GalEl sum1, sum2;
  GalEl Y;

  if ( nu >= dd )
    {
      // word is undecodable !
      return 1;
    }

  PSHOW(1<<6,"C1: X",X);
  for (j=1; j<=nu; j++)
    for (i=1; i<nu; i++)
      sigmaji[j][i] = Gadd(L[i],Gmul(X[j],sigmaji[j][i-1]));

  memcpy(vcorr,r,nn);
  for (j=1; j<=nu; j++)
    {
      sum1 = sum2 = ZERO;
      for (l=0; l<nu; l++) sum1 = Gadd(sum1,Gmul(sigmaji[j][l],E[nu-l]));
      for (l=0; l<nu; l++)
        sum2 = Gadd(sum2,Gmul(sigmaji[j][l],Gpow(X[j],nu-l)));
      Y = Gdiv(sum1, sum2);
      vcorr[X[j]] = Gadd(r[X[j]], Y);
    }
  return 0;
}
```

```
void C1Decoder::setoutputeraflags(int i)
{
  memset(eraflags,i,FC2SZB);
}

int C1Decoder::geterasure(int i)
{
  return eraflags[i+nn-kk];
}



/*
 * C2.c
 * Copyright Kay Rydyger
 *
 * C2 decoder
 *
 */

#include "defs.h"
#include "classes.h"

static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

C2Decoder::C2Decoder(int decoder_strat)
{
  int i;
  for (i=0; i<nn; i++) {
    r[i] = ZERO;
    eraflags[i] = 0;
    sigmaji[i][0] = 0;
  }
  if (decoder_strat==2) do_decode=&C2Decoder::do_decode_strat2;
  if (decoder_strat==4) do_decode=&C2Decoder::do_decode_strat4;
}

void C2Decoder::C2Decode(union C2Frame *c2, union C2Frame *eraframe)
{
  int i;
  for (i=12;i<16;i++)
    {
      eraflags[i-12] = eraframe->byte[i];
      r[i-12] = c2->byte[i];    // Parity bits copied
    }
  for (i=0; i<12; i++)
    {
      eraflags[i+nn-kk] = eraframe->byte[i];
      r[i+nn-kk] = c2->byte[i];
    }
  for (i=16; i<FC2SZB; i++)
    {
      eraflags[i] = eraframe->byte[i];
      r[i] = c2->byte[i];
    }

  INFO(2,("C2 decoding Frame #:%d\n",frame));
  PSHOW(1<<11,"C2: erasures",eraflags);
  PSHOW(1<<7,"decode C2:",r);
  (this->*do_decode)();
  PSHOW(1<<7,"..decode C2 corr:",vcorr);
  for (i=0; i<12; i++)
    {
      eraframe->byte[i] = eraflags[i+nn-kk];
      c2->byte[i] = vcorr[i+nn-kk];
    }
  for (i=12; i<FC2SZB-(nn-kk); i++)
    {
      eraframe->byte[i+nn-kk] = eraflags[i+nn-kk];
      c2->byte[i+nn-kk] = vcorr[i+nn-kk];
    }
  PSHOW(1<<11,"C2: after decoding erasures",eraflags);
}
```

```
// ------------  protected member functions -----------


/*
  these strategies found in "Tehranchi: Reliability estimates for data recovered
  from compact discs" and
  "The fourth international conference on video and data recording, Southampton"
*/

void C2Decoder::do_decode_strat2() {

  GalEl E[nn+1];       // error correcting polynomial in frequency domain
  GalEl L[nn];         // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu, nuoferas;

  ret = get_syndromes1_2t(r,E);
  if ( ret == 0 )      //  zero error syndrome
    {
      INFO(2,("C2:zero error syndrome\n"));
      memcpy(vcorr,r,nn);
      setoutputeraflags(0);
      return;
    }
  berlekamp(E,L,noeraflags);
  PSHOW(1<<6,"C2:error locator polynomial L:",L);
  chien_search(L,loc);
  for(i=0,nu=1;i<nn;i++) if ( loc[i] == ZERO ) X[nu++] = i;
  nu--;

  if ( nu == 1 ) // single error syndrome
    {
      INFO(2,("C2:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      setoutputeraflags(0);
      return;
    }
  else
    {
      INFO(2,("C2:multiple error syndrome nu>1 (nu=%d)\n",nu));
      nuoferas=0;
      PSHOW(1<<6,"eraflags",eraflags);
      for (i=0;i<nn;i++) if (eraflags[i]) nuoferas++;
      if ( nuoferas > 2 )
        {
          INFO(2,("C2:number of erasures > 2 (nuoferas=%d), not correcting!\n",nuoferas));
          // copy  output erasures from input erasures !
          // done. eraflags are used after return.
          memcpy(vcorr,r,nn);
          return;
        }
      if ( nuoferas == 2 ) // try two erasure decoding
        {
          INFO(2,("C2:trying two erasure decoding, nuoferas=2\n"));
          ret = berlekamp(E,L,eraflags);
          if ( !ret && !correct_with_fourney(E,L) )
            {
              setoutputeraflags(0);
              INFO(2,("C2: corrected, no output erasures!\n"));
              return;
            }
        }
      INFO(2,("C2: number of erasures < 2 || (nuoferas==2 && (berlekamp or fourney:uncorrectable)), nuoferas=%d
      // assign output erasures to all symbols!
      setoutputeraflags(1);
      memcpy(vcorr,r,nn);
      return;
    }
}
```

```
void C2Decoder::do_decode_strat2old() {

  GalEl E[nn+1];        // error correcting polynomial in frequency domain
  GalEl L[nn];          // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu, nuoferas;

  ret = get_syndromes1_2t(r,E);
  if ( ret == 0 )       //  zero error syndrome
    {
      INFO(2,("C2:zero error syndrome\n"));
      memcpy(vcorr,r,nn);
      return;
    }
  berlekamp(E,L,noeraflags);
  /*
    in general: firstly do berlekamp without input erasures!
    because: Codeword might be right
  */
  PSHOW(1<<6,"C2:error locator polynomial L:",L);
  chien_search(L,loc);
  for(i=0,nu=1;i<nn;i++) if ( loc[i] == ZERO ) X[nu++] = i;
  nu--;


  if ( nu == 1 ) // single error syndrome
    {
      INFO(2,("C2:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      return;
    }
  else
    {
      INFO(2,("C2:multiple error syndrome nu>1 (nu=%d)\n",nu));
      nuoferas=0;
      PSHOW(1<<6,"eraflags",eraflags);
      for (i=0;i<nn;i++) if (eraflags[i]) nuoferas++;
      if ( nuoferas == 2 ) // try two erasure decoding
        {
          INFO(2,("C2:trying two erasure decoding, nuoferas=2\n"));
          berlekamp(E,L,eraflags);
          correct_with_fourney(E,L);
          return;
        }
      INFO(2,("C2: found more than two erasures, nuoferas=%d\n",nuoferas));
      memcpy(vcorr,r,nn);
    }
}

// Superstrategy

void C2Decoder::do_decode_strat5() {

  GalEl E[nn+1];        // error correcting polynomial in frequency domain
  GalEl L[nn];          // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu, nuoferas, eraAerr;

  ret = get_syndromes1_2t(r,E);
  if ( ret == 0 )       //  zero error syndrome
    {
      INFO(2,("C2:zero error syndrome\n"));
      memcpy(vcorr,r,nn);
      return;
    }
  berlekamp(E,L,noeraflags);
  PSHOW(1<<6,"C2:error locator polynomial L:",L);
  chien_search(L,loc);
  for(i=0,nu=1;i<nn;i++) if ( loc[i] == ZERO ) X[nu++] = i;
  PSHOW(1<<6,"C2:X",X);
  nu--;
```

```
if ( nu == 1 ) // single error syndrome
  {
    INFO(2,("C2:single error syndrome nu=1\n"));
    Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
    memcpy(vcorr,r,nn);
    vcorr[X[1]] = Gadd(r[X[1]], Y);
    return;
  }

nuoferas=0;
for (i=0;i<nn;i++) if (eraflags[i]) nuoferas++;
INFO(2,("C2: multiple error syndrome nu>1 (nu=%d)\n",nu));
if ( nuoferas <=4 )
  {
    INFO(2,("C2: number of erased symbols <= 4 (nuoferas=%d)\n",nuoferas));
    eraAerr=0;
    PSHOW(1<<6,"eraflags",eraflags);
    for (i=1;i<=nu;i++) if ( eraflags[X[i]] == 1 ) eraAerr++;

    if ( nu == 2 && eraAerr == 2 )
      {
        INFO(2,("C2:double error syndrome and V = 2\n"));
        INFO(2,\
             ("C2:trying two erasure decoding (modify two symbols)\n"));
        berlekamp(E,L,eraflags);
        correct_with_fourney(E,L);
        return;
      }
    else {
      INFO(2,("C2: not double error syndrom && V = 2 (nu=%d , V=%d)\n",\
              nu,eraAerr));
      if ( ( nu == 2 && (( eraAerr == 1 && nuoferas <= 3) ||
                         ( eraAerr == 0 && nuoferas<=  2) )) ||
           ( nuoferas <= 2  && nu != 2 ))
        {
          // assign erasure flags
          // to all symbols of the received word
          INFO(2,("C2: assign erasure flags to all symbols\n"));
        }
      else
        {
          // copy C2 erasure flags from C1 erasure flags
          INFO(2,("C2: copy C2 erasure flags to C1\n"));
        }
    }
  }
 else
   {
     INFO(2,("C2: number of erased symbols > 4\n"));
     // copy C2 erasure flags from C1 erasure flags
   }
 memcpy(vcorr,r,nn);
}

// ******************************STARTEGY 4**********************

// same as STRATEGY1
void C2Decoder::do_decode_strat4() {

  GalEl E[nn+1];       // error correcting polynomial in frequency domain
  GalEl L[nn];         // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu, nuoferas;

  ret = get_syndromes1_2t(r,E);
  if ( ret == 0 )      //  zero error syndrome
    {
      INFO(2,("C2:zero error syndrome\n"));
      memcpy(vcorr,r,nn);
      setoutputeraflags(0);
      return;
    }
```

```
berlekamp(E,L,noeraflags);
PSHOW(1<<6,"C2:error locator polynomial L:",L);
chien_search(L,loc);
for(i=0,nu=1;i<nn;i++) if ( loc[i] == ZERO ) X[nu++] = i;
nu--;

if ( nu == 1 ) // single error syndrome
  {
    INFO(2,("C2:single error syndrome nu=1\n"));
    Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
    memcpy(vcorr,r,nn);
    vcorr[X[1]] = Gadd(r[X[1]], Y);
    setoutputeraflags(0);
    return;
  }
else
  {
    INFO(2,("C2:multiple error syndrome nu>1 (nu=%d)\n",nu));
    nuoferas=0;
    PSHOW(1<<6,"eraflags",eraflags);
    for (i=0;i<nn;i++) if (eraflags[i]) nuoferas++;
    if ( nuoferas > 2 )
      {
        INFO(2,("C2:number of erasures > 2 (nuoferas=%d), not correcting!\n",nuoferas));
        // copy  output erasures from input erasures !
        // done. eraflags are used after return.
        memcpy(vcorr,r,nn);
        return;
      }
    if ( nuoferas == 2 ) // try two erasure decoding
      {
        INFO(2,("C2:trying two erasure decoding, nuoferas=2\n"));
        ret = berlekamp(E,L,eraflags);
        if ( !ret && !correct_with_fourney(E,L) )
          {
            setoutputeraflags(0);
            INFO(2,("C2: corrected, no output erasures!\n"));
            return;
          }
      }
    INFO(2,("C2: number of erasures < 2 || (nuoferas==2 && (berlekamp or fourney:uncorrectable)), nuoferas=%d
    // assign output erasures to all symbols!
    setoutputeraflags(1);
    memcpy(vcorr,r,nn);
    return;
  }
}

//*****************************startergy 4 END ********************************8

void C2Decoder::chien_search(GalEl *FTin, GalEl *out)
{
  GalEl  sum;
  int i,k;

  sum = FTin[nn-1];
  for( k=nn-2; k>=0; k-- ) sum = Gadd(sum,FTin[k]);
  out[0] = sum; // since inv(a0) = a0 ! ( and not a7 like below !)
  for( i=1; i<nn; i++ )
    {
      sum = FTin[nn-1];
      for( k=nn-2; k>=0; k-- ) sum = Gadd(Gmul(sum,ZERO-i),FTin[k]);
      out[i] = sum;
    }
  PSHOW(1<<6,"chien: loc",out);

}

int C2Decoder::correct_with_fourney(GalEl *E, GalEl *L)
{
  /* according to Peterson, Weldon: "Error correcting codes" p.297
   */
  int i,j,l,nu=1;
```

```
  GalEl sum1, sum2;
  GalEl Y, X[nn], loc[nn];

  chien_search(L,loc);
  PSHOW(1<<6,"C2:locator polynomial loc:",loc);
  /*
     zeros at 1 give the positions of the errors !!!
  */

  for (j=0,nu=1; j<nn; j++) if ( loc[j]==ZERO )  X[nu++] = j;
  nu--;
  if ( nu >= dd )
    {
      return 1;
    }

  PSHOW(1<<6,"C2:X",X);
  for (j=1; j<=nu; j++)
    for (i=1; i<nu; i++)
      sigmaji[j][i] = Gadd(L[i],Gmul(X[j],sigmaji[j][i-1]));
  memcpy(vcorr,r,nn);
  for (j=1; j<=nu; j++)
    {
      sum1 = sum2 = ZERO;
      for (l=0; l<nu; l++) sum1 = Gadd(sum1,Gmul(sigmaji[j][l],E[nu-l]));
      for (l=0; l<nu; l++)
        sum2 = Gadd(sum2,Gmul(sigmaji[j][l],Gpow(X[j],nu-l)));
      Y = Gdiv(sum1, sum2);
      vcorr[X[j]] = Gadd(r[X[j]], Y);
    }
  return 0;
}

void C2Decoder::do_decode_stratmy() {

  GalEl E[nn+1];        // error correcting polynomial in frequency domain
  GalEl L[nn];          // error locating polynomial
  GalEl Y, X[nn], loc[nn];
  int ret, i, nu, nuoferas, eraAerr;

  ret = get_syndromes1_2t(r,E);
  if ( ret == 0 )       //  zero error syndrome
    {
      INFO(2,("C2:zero error syndrome\n"));
      memcpy(vcorr,r,nn);
      return;
    }
  if ( berlekamp(E,L,noeraflags) )
    {
      INFO(2,("C2: according to first berlekamp without erasures : word is undecodable\n"));
    }


  PSHOW(1<<6,"C2:error locator polynomial L:",L);
  chien_search(L,loc);

  for(i=0,nu=1;i<nn;i++)
    if ( loc[i] == ZERO ) X[nu++] = i;
  PSHOW(1<<6,"C2:X",X);
  nu--;

  if ( nu == 1 ) // single error syndrome
    {
      INFO(2,("C2:single error syndrome nu=1\n"));
      Y = Gdiv(Gmul(sigmaji[1][0],E[1]),Gmul(sigmaji[1][0],X[1]));
      memcpy(vcorr,r,nn);
      vcorr[X[1]] = Gadd(r[X[1]], Y);
      return;
    }

  nuoferas=0;
  for (i=0;i<nn;i++) if (eraflags[i]) nuoferas++;
  INFO(2,("C2: multiple error syndrome nu != 1 (nu=%d)\n",nu));
```

```
if ( nuoferas <=4 )
  {
    INFO(2,("C2: number of erased symbols <= 4 (nuoferas=%d)\n",nuoferas));
    eraAerr=0;
    PSHOW(1<<6,"eraflags",eraflags);
    for (i=1;i<=nu;i++) if ( eraflags[X[i]] == 1 ) eraAerr++;

    if ( nu == 2 && eraAerr == 2 )
      {
        INFO(2,("C2:double error syndrome and V = 2\n"));
        INFO(2,\
             ("C2:trying two erasure decoding (modify two symbols)\n"));
        if ( berlekamp(E,L,eraflags) || correct_with_fourney(E,L) )
          {
            INFO(2,("C2: according to berlekamp or fourney: word is undecodable, no correction made \n"));

            memcpy(vcorr,r,nn);
          }
        return;
      }
    else
      {
        INFO(2,("C2: not double error syndrom && V = 2 (nu=%d , V=%d), trying erasure decoding!\n",nu,eraAerr

        if ( berlekamp(E,L,eraflags) || correct_with_fourney(E,L) )
                              /*
                                this step is additional to SUPERSTRATEGY and gives EC a try,
                                because the case nu=4 is not covered by the previous condition
                                and can be corrected with the help of erasures though
                              */
          {
            INFO(2,("C2: according to berlekamp or fourney: word is undecodable, no correction made \n"));

            memcpy(vcorr,r,nn);
          }


        INFO(2,("C2: not double error syndrom && V = 2 (nu=%d , V=%d)\n",\
                nu,eraAerr));
        if ( ( nu == 2 && (( eraAerr == 1 && nuoferas <= 3) ||
                           ( eraAerr == 0 && nuoferas <= 2) )) ||
             ( nuoferas <= 2  && nu != 2 ))
          {
                              /*
                                assign erasure flags
                                to all symbols of the received word
                              */
            INFO(2,("C2: assign erasure flags to all symbols\n"));
          }
        else
          {
            // copy C2 erasure flags from C1 erasure flags
            INFO(2,("C2: copy C2 erasure flags to C1\n"));
          }
      }
  }
 else
  {
    INFO(2,("C2: number of erased symbols > 4 (=%d, frame=%d)\n", nuoferas,frame));
    // copy C2 erasure flags from C1 erasure flags
    memcpy(vcorr,r,nn);
  }
 //        memcpy(vcorr,r,nn);


}


void C2Decoder::setoutputeraflags(int i)
{
  memset(eraflags,i,FC2SZB);
}
```

```
/*
 * RSDecoder.c
 * Copyright Kay Rydyger
 *
 * basic routines for RS decoder
 *
 */


#include "defs.h"
#include "classes.h"

static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";


GalEl RSDecoder::noeraflags[nn];

RSDecoder::RSDecoder()
{
        int i;
        for (i=0; i<nn; i++) noeraflags[i]=0;
}

int RSDecoder::get_syndromes1_2t(GalEl *r, GalEl *E) {
            /*
              slightly faster for (255,223) g++ -O6 than
              routine below (1m16<>1m12).
            */

        GalEl  sum;
        int i,k, decode=0;

        PSHOW(1<<6,"get_syndromes: r",r);

        for( k=1; k<=2*tt; k++)
        {
                sum = r[nn-1];
                for (i=nn-2; i>=0; i--) sum = Gadd(Gmul(sum,k),r[i]);
                E[k] = sum;
                if ( sum != ZERO ) decode++;
        }
        PSHOW(1<<6,"get_syndromes: E (1..2t)", E);
        return decode;
}

int RSDecoder::berlekamp(GalEl *S, GalEl *L, GalEl *eraflags) {
             /*
                according to "error-control techniques for digital communications"
                with computing of erasure polynomial in first loop (Blahut)
             */

         int i, l=0, n=1, degT=0, ss=0;

         GalEl delta;
         GalEl D[nn];
         GalEl T[nn];
         GalEl U[nn];
         PSHOW(1<<6,"erasures in berlekamp:",eraflags);

         ss=0;
         for (i=0;i<nn;i++)
           {
                 if (eraflags[i]) U[++ss]=i;
                 D[i] = ZERO; L[i] = ZERO;
         }
         if ( ss > dd-1 )
         {
 //                fprintf(stderr," ** ss=%d dd=%d berlekamp: word is undecodable (s>d-1) **\n",ss,dd);
                return 1;
         }

         D[0]=0; L[0]=0;
```

```
        while ( n <= ss )
        {
                l++;
                for ( i=nn-1; i>0; i-- )
                        D[i] = L[i] = Gadd( L[i], Gmul(L[i-1],U[l]));
                D[0] = L[0];
                n++;
        }
        while ( n < dd ) {
                delta = ZERO;
                for ( i=1; i<=l; i++ ) delta = Gadd( delta, Gmul( L[i], S[n-i]));
                delta = Gadd(delta, S[n]);
                Pmulz(D);
                if ( delta != ZERO )
                {
                        for ( i=0; i<nn; i++ )
                        {
                                T[i] = Gadd(Gmul(D[i],delta),L[i]);
                                if ( T[i] != ZERO ) degT = i;
                        }
                        if ( (l<<1) < n + ss )
                        {
                                l = n - l + ss;
                                if (delta==0)
                                {
                                        memcpy(D,L,nn*sizeof(GalEl));
                                }
                                else for(i=0;i<nn;i++)
                                        D[i] = Gmul(L[i],ZERO-delta);

                        }
                        memcpy(L,T,nn*sizeof(GalEl));
                }
                n++;
        }
        return 0;
}

/*
 * RSEncoder.c
 * Copyright Kay Rydyger
 *
 * basic routines for RS encoder
 *
 */


#include "defs.h"
#include "classes.h"

static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

RSEncoder::RSEncoder()
{
        int i;
        for (i=0; i<nn; i++) {
                vC1[i] = ZERO;
                vC2[i] = ZERO;
        }
}

void RSEncoder::RSEncode(union C2Frame *c2)
{
        int i;
        GalEl *data;
        data = vC2+nn-kk;
        for (i=0; i<nn-kk; i++) vC2[i] = ZERO ;
        for (i=0; i<12; i++) data[i] = c2->byte[i];
        for (i=16; i<FC2SZB; i++) data[i-nn+kk] = c2->byte[i];
        PSHOW(1<<8,"RSENCODE C2:  ",vC2);
        do_encode(vC2,data);
        for(i=0; i<nn-kk; i++) c2->byte[i+12] = vC2[i];
```

```
        PSHOW(1<<8,"RSENCODE C2 + parity:  ",vC2);
}

void RSEncoder::RSEncode(union C1Frame *c1)
{
        int i;
        GalEl *data;
        data = vC1+nn-kk;
        for (i=0; i<nn-kk; i++) vC1[i] = ZERO ;
        for (i=0; i<FC2SZB; i++) data[i] = c1->byte[i];
        PSHOW(1<<8,"RSENCODE C1:  ",vC1);
        do_encode(vC1,data);
        for(i=0; i<nn-kk; i++) c1->byte[i+28] = vC1[i];
        PSHOW(1<<8,"RSENCODE C1 + parity:  ",vC1);
}

// ------ protected members ---------

void RSEncoder::do_encode(GalEl *v, GalEl *data)
{
/*
   encoding according to Lin/Costello p.91.
   parity bits into first nn-kk bytes of codeword v,
   data is written into last kk bytes of v
*/
        int i,j;
        GalEl feedback;

        for (i=kk-1; i>=0; i--)
        {
                feedback = Gadd(data[i],v[nn-kk-1]);
                for (j=nn-kk-1; j>0; j--)
                        v[j] = Gadd(v[j-1],Gmul(gen[j],feedback));
                v[0] = Gmul(feedback,gen[0]);
        }
}

/*
 * RS.c
 * Copyright Kay Rydyger
 *
 * basic routines for Galois-field arithmetic
 *
 */


#include "defs.h"
#include "classes.h"

static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

RS::RS()
{
        init_lookuptables();
        generator_polynomial();
}

/* --------------- protected members only ------------- */

GalEl RS::lookpoly[qq], RS::lookzech[qq], RS::lookpower[qq], RS::gen[nn-kk+1];
unsigned int RS::polynom = 361;//285;    // 100011101(d285) => X8+X4+X3+X2+1  451(X8+X7+X6+X+1)


void RS::init_lookuptables(void) {

  unsigned int negmask = ~ZERO, oldvec;
  unsigned int poly = polynom & ZERO;
  int i;

  lookpoly[mask]=0;
  lookpoly[mm]=poly;

  oldvec = poly<<1;
```

```
  for ( i=0; i<mm; i++ ) lookpoly[i]=1<<i;
  for ( i=mm+1; i<nn; oldvec=(lookpoly[i]<<1),i++ )
    lookpoly[i] = (oldvec & negmask)? (oldvec ^ poly) & mask: oldvec;

  for ( i=0; i<nn; i++ ) lookpower[lookpoly[i]] = i;
  lookpower[0] = mask;

  for ( i=1; i<nn; i++ ) lookzech[i] = lookpower[lookpoly[i] ^ 1];
}

void RS::generator_polynomial(void) {

        int i,j;
        gen[0]=1; gen[1]=0;
        for (i=2; i<=nn-kk; i++) {
                gen[i] = 0;
                for (j=i-1; j>0; j--) gen[j] = Gadd(gen[j-1],Gmul(gen[j],i));
                gen[0] = Gmul(gen[0],i);
        }
}

GalEl RS::Gdiv(GalEl a,GalEl b)
{
        if ( b == 0 ) return a;
        if ( b == ZERO ) {
                printf("RS::Gdiv: internal error: *** division by ZERO ***\n");
                return ZERO; // wrong value returned !
        }
        return Gmul(a,ZERO-b);
}

GalEl RS::Gmul(GalEl a, GalEl b) {
        if (a==ZERO || b==ZERO) return ZERO;
        return (a+b)%nn;
}

GalEl RS::Gadd(GalEl a, GalEl b) {
        int ret;
        if (a==b) return ZERO;
        if (a==ZERO) return b;
        if (b==mask) return a;

        if (b<a) ret = lookzech[a-b] + b;
        else ret = lookzech[nn-b+a] + b;

        return ret%nn;
}

GalEl RS::Gpow(GalEl a, int b) {
        if (a==ZERO) return ZERO;
        return ((a*b)%nn);
}

void RS::Pmulz(GalEl *res) {

        int i;
        for ( i=nn-1; i>0; i-- ) res[i] = res[i-1];
        res[0]=ZERO;
}

void RS::Pshow(char *s, GalEl *a) {

        int i;
        printf("\nPolynomial: %s ",s);
        for (i=0; i<nn; i++) printf("%02X(%02X), ",a[i],i);
        puts("\n");
}


/*
 * GPC.cc
 * Copyright Kay Rydyger
 *
```

```
 * routines for simulating scratches etc.
 *
 */

#include "defs.h"
#include "classes.h"

static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

GPC::GPC(void)
{}

GPC::~GPC(void)
{}

int GPC::radial_error(int symbcnt)
{
  static int ges_circum = 660; //=min_circum
  static int add_circum=33;
  static int length_in_trs=10000;
  static int act_circum = 660; // =min_circum
  static int tr=0;

  if (symbcnt % ges_circum == 0 && tr++ < length_in_trs)
    {
      act_circum += add_circum;
      ges_circum += act_circum;
      return 1; // "scratch" encountered !
    }
  return 0;
}

int GPC::tangential_error(int symbcnt)
{
        return 0;
}

int GPC::is_bad(int symbcnt)
{
        if (radial_error(symbcnt)) return 1;
        if (tangential_error(symbcnt)) return 1;
        return 0;
}

int GPC::burst_gpc_length(Random *rnd, int wide)
  /* return burst length between 1 and wide-1 inclusive */
{
  int zufall;
  //  while ( (zufall = rnd->rnd_rand(wide)) == 0 );
  return wide;//zufall;
}


int GPC::gap_gpc_length(Random *rnd, int thinning)
{
  static int a=1;
  static int dist = 33; // war 20
  if (++a%2==0) return thinning;
  else return dist-thinning;
}


/*
 * Random.cc
 * Copyright Kay Rydyger
 *
 * routines for random numbers and error distributions
 *
 */

#include "defs.h"
#include "classes.h"
```

```
static char cvsid[]="$Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $";

//#define LINUX

Random::Random()
{
        idum=-1;
        ran1();

#ifdef LINUX
        idum = get_rnd_int();
#else
        idum=2;
#endif
        idumfast=0;
}

Random::~Random()
{}

#ifdef  LINUX

unsigned int Random::get_rnd_int()
{
        int fd,count,ran;

        fd = open(RANDOM_DEVICE,O_RDONLY);
        if (fd == 0) {
                printf("get_rnd_int():error opening %s\n",RANDOM_DEVICE);
                exit(1);
        }
        do
        {
                count = read(fd,&ran,4);
        } while (count != 4);
        if (close(fd) < 0) printf("get_rnd_int():error closing %s\n",RANDOM_DEVICE);
        return ran;
}
#endif

float Random::ran1(void)
{
  /*
     return a uniform deviate between 0.0 and 1.0 (exclusive of the endpoint value)
  */
        int j;
        long k;
        static long iy=0;
        static long iv[NTAB];
        float temp;


        if (idum <= 0 || !iy)
        {
                if (-(idum) < 1) idum=1;
                else idum = -(idum);
                for (j=NTAB+7; j>=0; j--)
                {
                        k = (idum)/IQ;
                        idum = IA * (idum - k*IQ) - IR*k;
                        if (idum < 0) idum += IM;
                        if (j < NTAB) iv[j] = idum;
                }
                iy = iv[0];
        }

        k=(idum)/IQ;
        idum = IA * (idum-k*IQ)-IR*k;
        if (idum < 0) idum += IM;
        j = iy/NDIV;
        iy = iv[j];
        iv[j] = idum;
        if ((temp=AM*iy) > RNMX) return RNMX;
```

```
        else return temp;
}

float Random::ranfast(void)
{
        unsigned long idum,itemp;
        float rand;
        static unsigned long jflone = 0x3f800000;
        static unsigned long jflmsk = 0x007fffff;
        idumfast = 1664525L * idumfast + 1013904223L;
        itemp = jflone | (jflmsk & idumfast);
        rand  = (*(float *)&itemp)-1.0;
        return rand;
}

// #define ran1 ranfast  //1.3 times faster


float Random::expdev(void)
{
        float dum;
        do
                dum = ran1();
        while (dum==0.0);
        return -log(dum);
}


float Random::gasdev(void)
{
        static int iset=0;
        static float gset;
        float fac,rsq,v1,v2;

        if (iset==0)
        {
                do
                {
                        v1 = 2.0*ran1()-1.0;
                        v2 = 2.0*ran1()-1.0;
                        rsq = v1*v1+v2*v2;
                } while (rsq >= 1.0 || rsq == 0.0);
                fac = sqrt(-2.0*log(rsq)/rsq);

                gset = v1*fac;
                iset=1;
                return v2*fac;
        }
        else {
                iset = 0;
                return gset;
        }
}

int Random::rnd_gauss(float limit1, float limit2, float spread, int shift)
{
        float random;
        int ran;
        do ran = (int) (gasdev() * spread + shift);
        while (ran <= limit1 || ran > limit2);
        return ran;
}

int Random::rnd_exp(void)
{
        int ran;
        do ran = (int) (expdev()*16);
        while (ran == 0);
        return ran;
}

int Random::rnd_flat(int width)
{
```

```
        int ran;
        do ran = (int) (ran1()*width);
        while (ran==0);
        return ran;
}

int Random::rnd_rand(int width)
{
  return (int) (ran1()*width);
}

int Random::rnd_randfast(int width)
{
  return (int) (ranfast()*width);
}

int Random::gap_rnd_length(float thinning)
{
  int zz;
  static int j=0;
  do {
    if (j++%35==0) zz = (int) (rnd_exp()*thinning);
    else zz = (int) (rnd_flat(10000)*thinning);
  } while (zz==0);
  return zz;
}

int Random::burst_rnd_length(void)
{
  int zz;
  static int j=0;
  do {
    j++;
    if (j%13==0) zz = (int) rnd_gauss(10,50,2,30);
    else if (j%1000==0) zz = (int) rnd_flat(300);
    else zz = (int) rnd_gauss(0,100,10,0);
  } while (zz==0);
  return zz;
}

int Random::gap_rnd_length3(float thinning)
{
  int zz;
  static int j=0;
  do {
    j++;
    if (ran1()<0.008) zz = (int) (expdev()*3);
    else zz = (int) (rnd_flat(10000)*thinning);
  } while (zz==0);
  return zz;
}

// statistics  from channelmodell
int Random::burst_rnd_length3(void)
{
  int zz;
  static int j=0;
  do {
    j++;
    zz = (int) (expdev()*1.1);
    if (j%1000==0) zz = (int) rnd_gauss(10,50,2,30);
    if (j%7000==0) zz = (int) (expdev()*20);
  } while (zz==0);
  return zz;
}

// statistics  from channelmodell  with a 10 times higher burst probability
int Random::burst_rnd_length4(void)
{
  int zz;
  static int j=0;
  do {
    j++;
```

```
    zz = (int) (expdev()*1.1);
    if (j%80==0) zz = (int) rnd_gauss(10,50,2,30);
    if (j%7000==0) zz = (int) (expdev()*20);
  } while (zz==0);
  return zz;
}

int Random::gilbert(void)
{
  static int state=1;
  float beta=0.8, alpha=0.1;
  if (state==1) {
    if (ran1()<beta)
      {
        return 1;
      }
    else {
      state=0;
      return 0;
    }
  }
  if (state==0) {
    if (ran1()<alpha) return 0;
    else {
      state=1;
      return 1;
    }
  }
  return 0;
}


/*
 * defs.h
 * Copyright Kay Rydyger
 *
 * defintions
 *
 */


// $Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $


/* includes */
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <limits.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>


/* general */
#define DEBUG 0//((1)|(2))//|(1<<11)|(1<<9)|(1<<6))

//#define EFMOFF
//#define NRZIOFF
//#define SCRAMBLEOFF // not working !
//#define RSOFF
#define FILLBYTE 10
#define BERFACTOR 100000

extern int frame;
extern int maxNonEFMWords;

#ifdef DEBUG
#define INFO(which,a) if (which&DEBUG) printf a;
#define PSHOW(which,a,b) if (/*frame<115 && */which&DEBUG) Pshow(a,b)
```

```
#define SHOWPUFFERC1(which,a,b) if (/*frame < 115 &&*/ which&DEBUG) {\
int i,j;\
        printf(a);\
        for(j=0;j<2;j++) {\
        for(i=0;i<FC1SZB;i++)\
printf("%02X ",b[j].byte[i]);\
cout << endl;}\
}

#define SHOWPUFFER_FSZB(which,a,b) if (/*frame < 115 &&*/ which&DEBUG) {\
int i,j;\
        printf(a);\
        for(j=0;j<3;j++) {\
        for(i=0;i<FSZB;i++)\
printf("%02X ",b[j].byte[i]);\
cout << endl;}\
}

#define SHOWPUFFER_FC2SZB(which,a,b) if (/*frame < 115 &&*/ which&DEBUG) {\
int i;\
printf(a);\
for(i=0;i<FC2SZB;i++)\
printf("%02X ",b->byte[i]);\
cout << endl;\
}

#define SHOWPUFFER_FSZB1(which,a,b) if (/*frame < 115 &&*/ which&DEBUG) {\
int i;\
printf(a);\
for(i=0;i<FSZB;i++)\
printf("%02X ",b.byte[i]);\
cout << endl;\
}

#define SHOWPUFFERC2(which,a,b) if (/*frame <115 && */which&DEBUG) {\
                    printf(a);\
                    for(j=0;j<112;j++) { \
                    printf("%03d: ",j); \
                    for(i=0;i<FC2SZB;i++) \
                    printf("%02X ",b[j].byte[i]); \
                    cout << endl; \
                    }\
                    }

#else
#define INFO(which,a) ;
#define PSHOW(which,a,b) ;
#define SHOWPUFFERC1(which,a,b) ;
#define SHOWPUFFERC2(which,a,b) ;
#endif

#define FSZB (sizeof(Frame)/sizeof(BYTE))
#define FSZW (sizeof(Frame)/sizeof(WORD))
#define FC1SZB (sizeof(C1Frame)/sizeof(BYTE))
#define FC1SZW (sizeof(C1Frame)/sizeof(WORD))
#define FC2SZB (sizeof(C2Frame)/sizeof(BYTE))
#define FC2SZW (sizeof(C2Frame)/sizeof(WORD))
#define FC1SZBCW (FC1SZB+1)

/* EFM */

#ifdef EFMOFF
#define TRAFOTOEFM
#define TRAFOFROMEFM
#else
#define TRAFOFROMEFM Efm->TrafoFromEFM
#define TRAFOTOEFM   Efm->TrafoToEFM
#endif

#ifdef NRZIOFF
#define TO_NRZI
#define FROM_NRZI
#else
#define TO_NRZI to_NRZI
```

```
#define FROM_NRZI from_NRZI
#endif

#define WRONGEFM 7
#define ft 14
#define maxft14 (1<<ft)
#define svteen ft+3

#define ddd 3
#define kkk 11
#define mask14  ((1<<ft)-1)
#define my_getbit(a,b) (a&(1<<b))
#define my_setbit(a,b)  (a|=(1<<b))
#define UP 1
#define DOWN -1
#define NOSUPPR 0
#define EFMTABLEFILE "./EFMTable"

/* Scrambler */

typedef unsigned char BYTE;
typedef unsigned short int WORD;

#define INC1R3(x) if (x<2) x++;else x=0;
#define INC1R2(x) if(x==0)x++;else x=0;

#define INC4R109(x) if (x<108) x+=4;else x-=108;
#define DEC4R109(y) if(y>3)y-=4;else y+=108;

#define INC1R109(x) if (x<111) x++;else x=0;
#define DEC1R109(y) if (y==0)y=111;else y--;

#ifdef RSOFF
#define RSENCODE(a) ;
#define C1DECODE(a,b) ;
#define C2DECODE(a,b) ;
#else
#define RSENCODE(a) RsEncoder->RSEncode(a)
#define C1DECODE(a,b) C1->C1Decode(a,b)
#define C2DECODE(a,b) C2->C2Decode(a,b)
#endif

/* RS encoder/decoder */

typedef unsigned char GalEl;
/*  (255,251) RS Code ok. */

#define mm 8
#define nn 255  // nn = 2^mm-1
#define tt 2
#define kk (nn - (tt<<1))
#define dd (2*tt + 1)
#define qq (1<<mm)
#define mask (qq-1)
#define ZERO mask


/* Randomizer */

#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 1277773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
#define RANDOM_DEVICE "/dev/random"
#define MAX_GAP_LENGTH 999999999


/*
```

```
* classes.h
* Copyright Kay Rydyger
*
* class declarations
*
*/


// $Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $


template <class arrclass> class Arr
{
  public:
        Arr(int sz) {
                int i;
                array = new arrclass(sz);
                for (i=0;i<sz;i++) array[i]=ZERO;
                size=sz;
                };
        arrclass &operator[](int i);
  protected:
        arrclass *array;
        int size;
};

template <class arrclass> arrclass &Arr<arrclass>::operator[](int i)
{
        if (i<0||i>=size) {
                printf("template arrclass: internal error: *** range checking error ***\n");
                printf("i=%d, size=%d ***\n",i,size);
                return ZERO;
        }
        return array[i];
}

struct Error_kinds {
  int single_errors;
  int interpolated_eras;
  int error_clicks;
  int errs_not_eras;
  int P00,P01,P10,P11;
  int bytes_checked;
};

union Sample {
        BYTE byte[4];
        WORD word[2];
};
union Frame {
        union Sample sample[6];
        BYTE byte[24];
        WORD word[12];
};
union C2Frame {
        union Sample sample[7];
        BYTE byte[28];
        WORD word[14];
};
union C1Frame {
        union Sample sample[8];
        BYTE byte[32];
        WORD word[16];
};

class Decode
{
  public:
        Decode(int);
        ~Decode();
        void oneFrame();
        int modified(void);
        int getChnbit(int);
```

```
        void setChnbit(int);
        void toggleChnbit(int);
        int *inputFrame;//[FC1SZBCW];
        union Frame outputFrame;
        union Frame outerasFrame;
        class EFM *Efm;
  protected:
        class C1Decoder  *C1;
        class C2Decoder  *C2;
        union C1Frame *EFMdecodeFrame();
        void DescrambleFrame(union C1Frame *input);


        union C1Frame *input;//[2];
        union C1Frame *inputera; //[2];
        union Frame *output;//[3];
        union Frame *outputera;//[3];
        union C2Frame *puffer;//[109];
        union C2Frame *erapuf; //[112];
        static union C2Frame eradumm;
        int Pos, Pos1, Pos2, PosBuf;
};

class Encode
{
  public:
        Encode();
        ~Encode();
        void oneFrame();
        int otherfill();
        int getChnbit(int);
        void setChnbit(int);
        void toggleChnbit(int);
        union Frame inputFrame;
        int *outputFrame;//[FC1SZB];
        class EFM *Efm;
        class RSEncoder  *RsEncoder;

  protected:

        void EFMencodeFrame(union C1Frame*);
        union C1Frame *ScrambleFrame();

        union C2Frame *input;//[3];
        union C1Frame *output;//[2];
        union C1Frame *puffer;//[112];
        int Pos1, Pos2, Pos, PosBuf, Pos3;
};

class EFM
{
  public:
        EFM();
        int TrafoToEFM(BYTE);
        int TrafoFromEFM(int);
        void printEFMTable(void);
        int getNonEFMWords(int i);
  protected:
        int EFMTable[256], EFMrevTable[maxft14];
        int NonEFMWordsTable[maxft14];
        unsigned DSV;
        static int MB[4];

        void CreateOfficialEFMTable();
        void CreateMyEFMTable(void);
        int CreateNonEFMWords(void);
        int test_constraints(int code);
        int to_NRZI(unsigned int EfmMb);
        int from_NRZI(int);
        int calcDSV(unsigned int efmword, int *d);
        int calc_aNulls(unsigned efmword);
        int calc_bNulls(unsigned efmword);
        int add_merging_bits(unsigned int *efmword);
```

```
};

class RS
{
  public:
        RS();
  protected:
        static GalEl lookpoly[qq], lookzech[qq], lookpower[qq], gen[nn-kk+1];
        static unsigned int polynom;

        void init_lookuptables(void);
        void generator_polynomial(void);
        GalEl Gdiv(GalEl, GalEl);
        GalEl Gmul(GalEl, GalEl);
        GalEl Gadd(GalEl, GalEl);
        GalEl Gpow(GalEl, int);
        void Pmulz(GalEl*);
        void Pshow(char *s, GalEl *a);

};

class RSEncoder : public RS
{
  public:
        RSEncoder();
        void RSEncode(union C2Frame*);
        void RSEncode(union C1Frame*);
  protected:
        GalEl vC1[nn],vC2[nn];
        void do_encode(GalEl*, GalEl*);
};

class RSDecoder : public RS
{
  public:
        RSDecoder();
  protected:
        static GalEl noeraflags[nn];
        int get_syndromes1_2t(GalEl*, GalEl*);
        int berlekamp(GalEl*, GalEl*, GalEl*);
};

class C1Decoder : public RSDecoder
{
  public:
        C1Decoder(int);
        void C1Decode(union C1Frame*,union C1Frame*);
        void setoutputeraflags(int);
        int geterasure(int);
        int assign_erasures;
  protected:
        GalEl eraflags[nn];
        GalEl sigmaji[nn+1][nn];
        GalEl r[nn], vcorr[nn];
        void (C1Decoder::*do_decode)(void);
        void do_decode_strat2();
        void do_decode_strat5();
        void do_decode_strat4();
        void do_decode_stratmy();
        int chien_search(GalEl*, GalEl*);
        int correct_with_fourney(GalEl*, GalEl*, GalEl*, int);
};

class C2Decoder : public RSDecoder
{
  public:
        C2Decoder(int);
        void C2Decode(union C2Frame*, union C2Frame*);
  protected:
        GalEl eraflags[nn];
        GalEl sigmaji[nn+1][nn];
        GalEl r[nn], vcorr[nn];
        void (C2Decoder::*do_decode)(void);
```

```
        void do_decode_strat2();
        void do_decode_strat5();
        void do_decode_strat4();
        void do_decode_stratmy();
        void do_decode_strat2old();
        void chien_search(GalEl*, GalEl*);
        int correct_with_fourney(GalEl*, GalEl*);
        void setoutputeraflags(int);
};

class Random
{
  public:
        Random(void);
        ~Random(void);
        int burst_rnd_length(void);
        int burst_rnd_length2(void);
        int burst_rnd_length3(void);
        int burst_rnd_length4(void);
        int gap_rnd_length(float);
        int gap_rnd_length3(float);
        float ranfast(void);
        int rnd_rand(int);
        int rnd_randfast(int);
        int gilbert(void);
  protected:
        unsigned int get_rnd_int(void);
        float ran1(void);
        float expdev(void);
        float gasdev(void);
        int rnd_gauss(float,float,float,int);
        int rnd_flat(int);
        int rnd_exp(void);
        long idum;
        unsigned long idumfast;

};

class GPC
{
  public:
        GPC(void);
        ~GPC(void);
        int is_bad(int);
        int gap_gpc_length(Random*,int);
        int burst_gpc_length(Random*,int);

  protected:
        int radial_error(int symbcnt);
        int tangential_error(int symbcnt);
};


/*
 * EFMTable.h
 * Copyright Kay Rydyger
 *
 * official EFM table
 *
 */


// $Id: listings.tex,v 1.4 2002/05/19 11:58:11 kay Exp $

char EFMTable_inc[][32]=
{
  "0    01001000100000",
  "1    10000100000000",
  "2    10010000100000",
  "3    10001000100000",
  "4    01000100000000",
  "5    00000100010000",
  "6    00010000100000",
```

```
"7    00100100000000",
"8    01001001000000",
"9    10000001000000",
"10   10010001000000",
"11   10001001000000",
"12   01000001000000",
"13   00000001000000",
"14   00010001000000",
"15   00100001000000",
"16   10000000100000",
"17   10000010000000",
"18   10010010000000",
"19   00100000100000",
"20   01000010000000",
"21   00000010000000",
"22   00010010000000",
"23   00100010000000",
"24   01001000010000",
"25   10000000010000",
"26   10010000010000",
"27   10001000010000",
"28   01000000010000",
"29   00001000010000",
"30   00010000010000",
"31   00100000010000",
"32   00000000100000",
"33   10000100001000",
"34   00001000100000",
"35   00100100100000",
"36   01000100001000",
"37   00000100001000",
"38   01000000100000",
"39   00100100001000",
"40   01001001001000",
"41   10000001001000",
"42   10010001001000",
"43   10001001001000",
"44   01000001001000",
"45   00000001001000",
"46   00010001001000",
"47   00100001001000",
"48   00000100000000",
"49   10000010001000",
"50   10010010001000",
"51   10000100010000",
"52   01000010001000",
"53   00000010001000",
"54   00010010001000",
"55   00100010001000",
"56   01001000001000",
"57   10000000001000",
"58   10010000001000",
"59   10001000001000",
"60   01000000001000",
"61   00001000001000",
"62   00010000001000",
"63   00100000001000",
"64   01001000100100",
"65   10000100100100",
"66   10010000100100",
"67   10001000100100",
"68   01000100100100",
"69   00000000100100",
"70   00010000100100",
"71   00100100100100",
"72   01001001000100",
"73   10000001000100",
"74   10010001000100",
"75   10001001000100",
"76   01000001000100",
"77   00000001000100",
"78   00010001000100",
"79   00100001000100",
```

```
"80   10000000100100",
"81   10000010000100",
"82   10010010000100",
"83   00100000100100",
"84   01000010000100",
"85   00000010000100",
"86   00010010000100",
"87   00100010000100",
"88   01001000000100",
"89   10000000000100",
"90   10010000000100",
"91   10001000000100",
"92   01000000000100",
"93   00001000000100",
"94   00010000000100",
"95   00100000000100",
"96   01001000100010",
"97   10000100100010",
"98   10010000100010",
"99   10001000100010",
"100  01000100100010",
"101  00000000100010",
"102  01000000100100",
"103  00100100100010",
"104  01001001000010",
"105  10000001000010",
"106  10010001000010",
"107  10001001000010",
"108  01000001000010",
"109  00000001000010",
"110  00010001000010",
"111  00100001000010",
"112  10000000100010",
"113  10000010000010",
"114  10010010000010",
"115  00100000100010",
"116  01000010000010",
"117  00000010000010",
"118  00010010000010",
"119  00100010000010",
"120  01001000000010",
"121  00001001001000",
"122  10010000000010",
"123  10001000000010",
"124  01000000000010",
"125  00001000000010",
"126  00010000000010",
"127  00100000000010",
"128  01001000100001",
"129  10000100100001",
"130  10010000100001",
"131  10001000100001",
"132  01000100100001",
"133  00000000100001",
"134  00010000100001",
"135  00100100100001",
"136  01001001000001",
"137  10000001000001",
"138  10010001000001",
"139  10001001000001",
"140  01000001000001",
"141  00000001000001",
"142  00010001000001",
"143  00100001000001",
"144  10000000100001",
"145  10000010000001",
"146  10010010000001",
"147  00100000100001",
"148  01000010000001",
"149  00000010000001",
"150  00010010000001",
"151  00100010000001",
"152  01001000000001",
```

```
"153  100000010010000",
"154  100100000000001",
"155  100010000000001",
"156  010000010010000",
"157  000010000000001",
"158  000100000000001",
"159  001000010010000",
"160  000010000100001",
"161  100000100001001",
"162  010000100010000",
"163  000001001000001",
"164  010001000001001",
"165  000001000001001",
"166  010000001000001",
"167  001001000001001",
"168  010010010010001",
"169  100000010010001",
"170  100100010010001",
"171  100010010010001",
"172  010000010010001",
"173  000000010010001",
"174  000100010010001",
"175  001000010010001",
"176  000001001000000",
"177  100000100010001",
"178  100100100010001",
"179  001001000100000",
"180  010000100010001",
"181  000000100010001",
"182  000100100010001",
"183  001000100010001",
"184  010010000010001",
"185  100000000010001",
"186  100100000010001",
"187  100010000010001",
"188  010000000010001",
"189  000010000010001",
"190  000100000010001",
"191  001000000010001",
"192  010001001000000",
"193  100001000100001",
"194  100100100100000",
"195  000010001001000",
"196  010001000100001",
"197  000001000100001",
"198  000100100100000",
"199  001001000100001",
"200  000010010000001",
"201  100001000000001",
"202  000010010001000",
"203  000010010000000",
"204  010001000000001",
"205  000001000000001",
"206  000000100100000",
"207  001001000000001",
"208  000001001001000",
"209  100000100100001",
"210  100100100100001",
"211  100001001000000",
"212  010000100100001",
"213  000000100100001",
"214  000100100100001",
"215  001000100100001",
"216  010010000100001",
"217  100000000100001",
"218  100100000100001",
"219  100010000100001",
"220  010000000100001",
"221  000010000100001",
"222  000100000100001",
"223  001000000100001",
"224  010001000000100",
"225  000001000000100",
```

```
  "226  10000100010010",
  "227  00100100000010",
  "228  01000100010010",
  "229  00000100010010",
  "230  01000000100010",
  "231  00100100010010",
  "232  10000100000010",
  "233  10000100000100",
  "234  00001001001001",
  "235  00001001000010",
  "236  01000100000100",
  "237  00000100000100",
  "238  00010000100010",
  "239  00100100000100",
  "240  00000100100010",
  "241  10000010010010",
  "242  10010010010010",
  "243  00001000100010",
  "244  01000010010010",
  "245  00000010010010",
  "246  00010010010010",
  "247  00100010010010",
  "248  01001000010010",
  "249  10000000010010",
  "250  10010000010010",
  "251  10001000010010",
  "252  01000000010010",
  "253  00001000010010",
  "254  00010000010010",
  "255  00100000010010"
};
```

References

[1] L. M. H. E. Driessen and L. B. Vries, eds., *Performance calculations of the Compact Disk error correction code on a memoryless channel*, Proceedings of the Fourth International Conference on Video and Data Recording, (Southhampton), University of Southhampton, 20-23 April 1982.

[2] B. Tehranchi and D. G. Howe, "A channel model for characterization of the error data recovered from compact discs," *IEEE Transactions on Communications*, vol. 46, pp. 841–845, July 1998.

[3] 2002 Reuters Limited, "CD-Erfinder Philips sieht keine Zukunft für Kopierschutz," *Financial Times Deutschland*, January, 9th 2002.

[4] R. Iannella, "Digital Rights Management (DRM) Architectures," *D-Lib Magazine*, vol. 7, June 2001.

[5] Philips Semiconductors, *Data sheet, SAA7325 Digital servo processor and compact disc decoder with integrated DAC(CD10)*, June 17 1999.

[6] "The compact disc digital audio system," in *British Standard 7064*, London: British Standard Institution, 1989.

[7] M. G. Carasso, J. B. H. Peek, and J. P. Sinjou, "The compact disc digital audio system," *Philips Tech. Rev.*, no. 40, pp. 151–155, 1982.

[8] Y. Sako and T. Suzuki, "Data structure of the compact disk-read-only memory system," *Applied Optics*, vol. 25, pp. 3996–4000, 15 Nov 1986.

[9] Y. Mitsuhashi, "Standardization activities for optical disks in japan," *Applied Optics*, vol. 25, pp. 4013–4016, 15 Nov 1986.

[10] F. A. Stevenson, "Cryptanalysis of contents scrambling system." November 1999.

[11] J. A. Bloom, I. J. Cox, T. Kalker, J.-P. M. G. Linnartz, M. L. Miller, and C. B. S. Traw, "Copy Protection for DVD Video," *Proceedings of the IEEE*, vol. 87, pp. 1267–1276, July 1999.

[12] O. Kastl, "CloneCD." http://www.elby.de/.

[13] S. Katzenbeisser and F. A. P. Petitcolas, eds., *Information hiding techniques for steganography and digital watermarking.* Artech House, 2000.

[14] R. J. Anderson and F. A. P. Petitcolas, "On the limits of steganography," *IEEE Journal of Selected Areas in Communications*, vol. 16, pp. 474–481, May 1998.

[15] J. Fridrich, "Image watermarking for tamper detection," *PROC. of IEEE Int. Conf. Image Processing*, 1998.

[16] M. W. Yeung and F. C. Mintzer, "Invisible watermarking for image verification," *Journal of Electronic Imaging*, vol. 7, no. 3, pp. 578–591, 1998.

[17] B. Schneier, *Applied Cryptography.* John Wiley & Sons, 1996.

[18] J. Lee and C. S. Won, "A watermarking sequence using parities of error control coding for image authentication and correction," *IEEE Transactions on Consumer Electronics*, vol. 46, pp. 313–317, May 2000.

[19] M. Y. Rhee, *Cryptography and Secure Communications*. McGraw-Hill, 1994.

[20] Basler AG Ahrensburg, *CD-Scanner*. sales brochure.

[21] ESP Laser Matrix, *ESP Inc.*, 2002. www.esp-cd.com.

[22] J. Benshop and G. van Rosmalen, "Confocal compact scanning optical microscope based on compact disc technology," *Applied Optics*, vol. 30, pp. 1179–1184, Apr 1991.

[23] "Volume and file structure of cdrom for information interchange," in *ECMA-119*, ECMA - Standardizing Information and Communication Systems, 2nd ed., December 1987.

[24] "Data interchange on read-only 120mm optical data disks (cd-rom)," in *ECMA-130*, ECMA - Standardizing Information and Communication Systems, 2nd ed., June 1996.

[25] "Audio recording - compact disc digital audio system," in *IEC 60908*, International Electrotechnical Commission, 2.0 ed., February 1999.

[26] Y. Sako and T. Suzuki, "Data structure of the compact disk-read-only memory system," *Applied Optics*, vol. 25, pp. 3996–4000, November 1986.

[27] J. D. Roberts, A. Ryley, D. M. Jones, and D. Burke, "Analysis of error correction constraints in an optical disk," *Applied Optics*, vol. 35, pp. 3915–3924, July 1996.

[28] L. B. Vries and K. Odaka, "CIRC-the error correcting code for the compact disc digital audio system," in *Collected papers from the AES Digital Audio Premier Conference*, pp. 178–186, Audio Engineering Society, 1982.

[29] B. Tehranchi and D. G. Howe, "Reliability estimates for data recovered from compact discs," *Applied Optics*, vol. 37, no. 2, 1998.

[30] Z. Yang, "Statistical reliability analysis of data recovered from compact discs," Master's thesis, Department of Electrical and computer engineering, University of Arizona, Tucson, Ariz., 1995.

[31] B. Tehranchi and D. G. Howe, "Error characteristics of read-only-memory versus write-once-read-many compact discs: CD-ROM versus CD-WORM," *Applied Optics*, vol. 35, no. 29, 1996.

[32] D. P. Casasent and A. G. Tescher, eds., *Real-Time Resolution, PC-Based System for Measurement or Errors on Compact Discs*, SPIE, 1994.

[33] C. C. Ko and T. T. Tjhung, "Comparison of simple cross-interleaved reed-solomon decoding strategies for compact disc players," in *Proceedings of the tenth international conference and industrial electronics and application TENCON 87*, pp. 378–382, 1987.

[34] E. Siever, S. Spainhour, J. P. Hekman, and S. Figgins, *Linux in a Nutshell.* O'Reilly, 3rd ed., 2000.

[35] *Principles of optical disc systems.* Adam Hilger Ltd., 1986.

[36] *GNU Compiler Collection.* http://www.gnu.org/software/gcc/gcc.html.

[37] American National Standard of Accredited Standards Committee X3, SMALL COMPUTER SYSTEM INTERFACE - 2 (SCSI-2) (draft), 9 March 1990.

[38] H. Ogawa and K. A. I. Schouhamer, "EFM - the modulation for the compact digital audio disc," in *Proc. AES Premier Conf., Ryetown*, pp. 117–124, New York: Audio Engineering Society, 1982.

[39] L. Baert, L. Theunissen, and G. Vergult, eds., *Digital Audio and Compact Disc Technology.* Newnes, 1992.

[40] L. B. Vries, K. A. I. Schouhamer, J.G.Nijboer, H. Hoeve, J. Timmermans, L. M. Driessen, T. Doi, K. Odaka, S. Furukawa, I. K. Iwamoto, Y. Sako, H. Ogawa, and T. Itoh, "The digital compact disc – modulation and error correction," in *J. Audio Eng. Soc. (Abstracts)*, vol. 28, 67th convention of the audio engineering society, Dec. 1980.

[41] Immink and et al., *Method of coding binary data.* United States Patent 4501000, 19. February 1985.

[42] K. A. I. Schouhamer and U. Gross, *On low frequence properties of EFM modulation.* Philips Research Laboratories Eindhoven - The Netherlands, 1982.

[43] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C.* Cambridge University Press, 1996.

[44] Lodewijk, B. Vries, K. A. Immink, J. G. Nijboer, and H. Hoeve, "The compact disc digital audio system: modulation and error-correction," in *67th Audio Engineering Society Convention*, no. 1674, (New York), 1981.

[45] H. Nakajima and H. Ogawa, *Compact Disc Technology.* Ohmsha Ltd., 1996.

[46] M. Godwin, "A Cop in Every Computer," *http://www.law.com*, 16. January 2002.

[47] T. T. Doi, "Error correction for digital audio recordings," in *Collected papers from the AES Digital Audio Premier Conference*, pp. 147–177, Audio Engineering Society, 1982.

[48] E. N. Gilbert, "Capacity of a burst-noise channel," *Bell Sys. Tech. J.*, vol. 39, pp. 1253–1263, 1960.

[49] T. T. Doi, Y. Tsuchiya, and A. Iga, "On several standards for converting pcm signals into video signals," *Journal of the Audio Eng. Soc. (Engineering Reports)*, vol. 26, pp. 641–649, 1978.

[50] Macrovision, *Safe Audio Product Overview*, 2002.

[51] S. Lin and J. Daniel J. Costello, *Error control coding: Fundamentals and applications.* Prentice Hall, 1983.

[52] E. R. Berlekamp, *Algebraic coding theory.* McGraw-Hill, 1968. New York.

[53] J. L. Massey, "Step-by-step decoding of the bose-chaudhuri-hocquenghem codes," *IEEE Trans. Inf. Theory*, vol. IT-11, pp. 580–585, October 1965.

[54] R. T. Chien, "Cyclic decoding procedure for the bose-chaudhuri-hocquenghem codes," *IEEE Trans. Inf. Theory*, vol. IT-10, pp. 357–363, october 1964.

[55] G. D. Fourney, "On decoding binary BCH codes," *IEEE Trans. Inf. Theory*, vol. IT-11, october 1965.

[56] A. M. Michelson and A. H. Levesque, *Error-control techniques for digital communication.* John Wiley & Sons, 1985.

[57] R. E. Blahut, *Theory and practice of error control codes.* Addison-Wesley Publishing Company, 1983.