

Implementation of a Watermarking Algorithm for H.264 Video Sequences

Examensarbete utfört i bildkodning
av

David Bergkvist


LiTH-ISY-EX-3516-2004
Linköping 2004

Implementation of a Watermarking Algorithm for H.264 Video Sequences

Thesis work performed in image coding
at Linköpings Tekniska Högskola
by
David Bergkvist

LiTH-ISY-EX-3516-2004

Supervisor: Karl-Göran Stenborg, Linköpings Tekniska Högskola
Examinator: Robert Forchheimer, Linköpings Tekniska Högskola
Linköping 2004-05-07

 LINKÖPING UNIVERSITET	Avdelning, Institution Division, Department	Datum Date 2004-05-07
	Institutionen för systemteknik 581 83 LINKÖPING	

Språk Language Svenska/Swedish X Engelska/English	Rapporttyp Report category Licentiatavhandling X Examensarbete C-uppsats D-uppsats Övrig rapport _____	ISBN ISRN LITH-ISY-EX-3516-2004 Serietitel och serienummer ISSN Title of series, numbering _____
URL för elektronisk version http://www.ep.liu.se/exjobb/isy/2004/3516/		

Titel Title	Implementation av en vattenmärkningsalgoritm för H.264-videosekvenser Implementation of a Watermarking Algorithm for H.264 Video Sequences
Författare Author	David Bergkvist

Sammanfattning Abstract <p>In today's video delivery and broadcast networks, issues of copyright protection have become more urgent than in analog times, since the copying of digital video does not result in the decrease in quality that occurs when analog video is copied. One method of copyright protection is to embed a digital code, a "watermark", into the video sequence. The watermark can then unambiguously identify the copyright holder of the video sequence. Watermarks can also be used to identify the purchaser of a video sequence, which is called "fingerprinting". The objective of this master thesis was to implement a program that would insert watermarks into video sequences and also detect if a given video sequence contains a given watermark. The video standard I chose to use was the H.264 standard (also known as MPEG4 AVC) as it offers a significant efficiency improvement over the previous video compression standards. A couple of tests that can be considered representative for most image manipulations and attacks were performed. The program passed all tests, suggesting that the watermarking mechanism of this thesis can be expected to be rather robust, at least for the video sequence used. By looking at the watermarked video sequences and comparing them to the originals, or measuring the signal to noise ratio, one can also see that the watermarks are unobtrusive. The execution times were also measured. Compared to coding and decoding a H.264 video stream, the time it takes to insert and extract watermarks was much less. Calculating a threshold takes roughly double the time as decoding the sequence, though.</p>

Nyckelord Keyword watermark, fingerprinting, image coding
--

Abstract

In today's video delivery and broadcast networks, issues of copyright protection have become more urgent than in analog times, since the copying of digital video does not result in the decrease in quality that occurs when analog video is copied.

One method of copyright protection is to embed a digital code, a "watermark", into the video sequence. The watermark can then unambiguously identify the copyright holder of the video sequence. Watermarks can also be used to identify the purchaser of a video sequence, which is called "fingerprinting".

The objective of this master thesis was to implement a program that would insert watermarks into video sequences and also detect if a given video sequence contains a given watermark.

The video standard I chose to use was the H.264 standard (also known as MPEG4 AVC) as it offers a significant efficiency improvement over the previous video compression standards.

A couple of tests that can be considered representative for most image manipulations and attacks were performed. The program passed all tests, suggesting that the watermarking mechanism of this thesis can be expected to be rather robust, at least for the video sequence used. By looking at the watermarked video sequences and comparing them to the originals, or measuring the signal to noise ratio, one can also see that the watermarks are unobtrusive. The execution times were also measured. Compared to coding and decoding a H.264 video stream, the time it takes to insert and extract watermarks was much less. Calculating a threshold takes roughly double the time as decoding the sequence, though.

Table of Contents

1	Introduction.....	1
1.1	Background.....	1
1.2	Purpose.....	1
1.3	Outline.....	1
1.4	Definitions.....	1
2	Theory.....	3
2.1	Video Coding Theory.....	3
2.1.1	Color Coding.....	3
2.1.2	Entropy Coding.....	3
2.1.3	Predictive Coding.....	3
2.1.4	Transform Coding.....	3
2.2	Watermark Theory.....	4
2.2.1	Watermark Properties.....	4
2.2.2	Inserting a Watermark.....	4
2.2.3	Comparing a Watermark.....	4
2.2.4	Generating a Threshold.....	5
2.2.5	Watermarks and Predictive Coding.....	5
3	The H.264 Standard.....	7
3.1	Sub Blocks.....	7
3.2	Macro Blocks.....	7
3.3	Slices and Frames.....	7
3.4	Parameter Sets.....	8
3.5	Network Abstraction Layer Units.....	8
4	Implementation.....	9
4.1	Generating a Watermark.....	9
4.2	Inserting a Watermark.....	9
4.3	Comparing a Watermark.....	10
4.4	Generating Threshold.....	11
4.5	Watermarks Sequences.....	11
4.6	Known Bugs and Limitations.....	11
5	Testing.....	13
5.1	Basic Usage.....	13
5.2	Watermarks Sequences.....	13
5.3	Recoding.....	14
5.4	Double Watermark Insertion.....	15
5.5	Collusion.....	15



5.5.1 Collusion by Swapping Macro Blocks.....	16
5.5.2 Collusion by Averaging.....	16
5.6 Unobtrusiveness.....	17
5.7 Time Consumption.....	18
6 Conclusion.....	19
7 References.....	21

1 Introduction

1.1 Background

In today's video delivery and broadcast networks, issues of copyright protection have become more urgent than in analog times, since the copying of digital video does not result in the decrease in quality that occurs when analog video is copied.

Cryptography can not be used to solve the problem, because once the data has been decrypted, there is no way to track or prevent it from being copied and distributed.

One method of copyright protection is to embed a digital code, a "watermark", into the video sequence. The watermark can then unambiguously identify the copyright holder of the video sequence.

Another, more useful, use of watermarks is to let the watermark identify the purchaser of the video sequence. If an illegally redistributed copy of the video sequence is discovered, it will then be possible to identify which of the purchasers that redistributed it, and either take legal action against the purchaser, or simply choose not to do business with him/her in the future. This use of watermarking is called "fingerprinting".

Both these uses of watermarking requires that the original non-watermarked video sequence is not released, because if the original non-watermarked video sequence is available, pirates distribute that one, and ignore the watermarked copies.

1.2 Purpose

The purpose of this master thesis was to implement and test a program that would:

- Insert a watermark sequence into a video sequence.
- Given an original, non-watermarked, video sequence v and a watermarked video sequence v^* and a watermark w , determine if it is likely that w is the same watermark as the one that was inserted into v to get v^* .

The program should support a common or upcoming video standard. There was no requirement for the program to support putting watermarks into audio sequences.

1.3 Outline

This master thesis contains the following chapters:

1. *Introduction* – In this chapter, the background and purpose of this master thesis is described, as well as definitions and a document outline (which is what you are reading now).
2. *Theory* – Describes what properties a watermark should have and how they should be designed to achieve those properties, as well as the principle on how watermarks are inserted and measured.
3. *The H.264 Video Standard* – Describes the parts of the H.264 video standard that are important to this master thesis.
4. *Implementation* – Describes how the program I implemented works and how to use it.
5. *Testing* – Describes what testing that was done, including tests on how pirates can attack the watermarks.
6. *Conclusions* – Discusses the testing results and if the program was successful or not.
7. *References* – Contains a list of the references that are used in this document

1.4 Definitions

The following definitions apply for this master thesis:

- *Amplitude* – A watermark's amplitude is the standard deviation of its coefficients.

- *Attacking* – A pirate attacks a watermark if he attempts to remove it, or fool the watermark program that the watermark is not there.
- *Chrominance* – The chrominance of a pixel describes the color of the pixel.
- *Collusion* – A collusion is when several pirates, who each have a copy of a video sequence with a unique watermark, combine their video sequences into one that only contains a very faint watermark.
- *False positive* – A false positive is an erroneous match.
- *Fingerprinting* – Fingerprinting means that there are several purchasers of a video sequence, and all get a uniquely watermarked version of the video sequence.
- *Frame* – A frame is a single image in a video sequence.
- *Luminance* – The luminance of a pixel describes the brightness of the pixel.
- *Match* – If a watermark comparison algorithm concludes that a given video sequence contains a given watermark, this is called a match. In this thesis, it will be called a match even if the comparison algorithm was wrong.
- *Pirate* – In this thesis, a pirate is someone who illegally redistributes a video sequence.
- *Signal to Noise Ratio* – The signal to noise ratio, in this thesis, is equal to the variation of a video sequence divided by the distortion that a watermark generated, measured in decibel.
- *Transform coefficient* – A frame in a transform coded video sequence is a sequence of numbers. These numbers are called transform coefficients.
- *Unobtrusive* – A watermark is unobtrusive if it is perceptually invisible.
- *Video sequence* – A video sequence is a sequence of frames. In this thesis, only digital video sequences are considered.
- *Watermark* – A watermark is a digital code that can be inserted into video sequences.
- *Watermark coefficient* – A watermark consist of a sequence of numbers. Each number is called a watermark coefficient.
- *Watermarked video sequence* – A watermarked video sequence is a video sequence with a watermark inserted into it.

2 Theory

This chapter describes the mathematical theory behind watermarks.

2.1 Video Coding Theory

Before we dive into the theory behind adding and comparing watermarks, it is useful to know how video coding is performed. Readers that are already well versed in this subject may want to skip this particular section and go directly to section 2.2.

2.1.1 Color Coding

The simplest way to represent colors is to divide each pixel into its three basic component values: red, green and blue. This is called RGB coding.

Another way to represent colors is what is called YUV coding, which means that each pixel is divided not into red, green and blue, but into three other components, namely one luminance component (the brightness of the pixel) and two chrominance components (the color of the pixel).

YUV coding does in itself not save any space. However, combined with the knowledge that the human visual system has a greater luminance resolution than chrominance resolution, it means that the chrominance frames can be desampled without anyone noticing. YUV 4:2:0 coding, which is popular in video coding, means that the chrominance frames have half height and width as the luminance frames.

2.1.2 Entropy Coding

Entropy coding means that the more likely a value (or sequence of values, if the values are correlated) is, the less bits are used to represent it. Conversely, the less likely a value (or sequence of values) is, the more bits are used to represent it.

2.1.3 Predictive Coding

Predictive coding means that instead of coding a value v_i , the difference between v_i and some prediction p_i is coded. The prediction is generally calculated as:

$$p_i = \sum(a_j \cdot v_{f(i)})$$

Where a_j are some constants and $v_{f(i)}$ are some previous values in the video sequence.

In video coding, there are generally whole frames that use predictive coding where the predictions are based on the previous frame.

2.1.4 Transform Coding

Transform coding means that the screen is divided into blocks of size $N \times N$, and a block is seen as a vector x of length N^2 . Instead of storing x directly, it is multiplied with a transform matrix T , and the result, $y = T \cdot x$, is stored instead.

The reason why this is done is that the pixel values in the video sequence are correlated, i.e. pixels close to each other have similar values. This means that storing the pixel values after each other (i.e. storing the x vector as it is), is a waste of space. If the transformation matrix T is chosen clever enough, the y values will be independent of each other.

Note that by itself, transform coding offer no compression, since both the x and the y vectors have the same length, and storing N^2 elements with a constant r bits per element takes up the same $r \cdot N^2$ bits regardless if the elements are independent of each other or not. However, when combined with entropy coding (see section 2.1.2), the y vector will take up less space than the x vector, since T can be chosen such that many y values become zero after quantization.

Obviously, in order to be able to extract x again, which is necessary to show the video sequence, T must be an invertable matrix.

2.2 Watermark Theory

A watermark is a digital code embedded into a video sequence. The watermark either uniquely identifies the copyright owner of the video sequence, or the purchaser of the video sequence. Using watermarks to identify the purchaser of a video sequence is called “fingerprinting”.

2.2.1 Watermark Properties

In order for a watermark to be effective, it should, according to [1], have the following properties:

- *Unobtrusive* – The watermark should be perceptually invisible.
- *Unambiguous* – Retrieval of the watermark should uniquely identify the copyright holder of the video sequence (or the purchaser in case “fingerprinting” is used).
- *Robust* – The watermark should either be impossible to remove or removing it should result in severe video degradation. Attempting to remove a watermark is referred to as “attacking” the watermark in this thesis.

In particular, a watermark should be robust to the following attacks:

- *Common signal processing* – The watermark should still be retrievable even if common signal processing operations, such as resampling, requantization, digital-to-analog and analog-to-digital conversions and common signal enhancements to image contrast or color.
- *Common geometric distortions* – The watermark should survive geometric operations such as rotation, translation, cropping and scaling.
- *Collusion and forging* – The watermark should be robust to collusion by multiple purchasers who each own a different watermarked copy of the video sequence. It should also be impossible for colluders to combine their video sequences to generate a different watermark with the intention of framing a third party.

Since most video standards use transform coding, and thus video sequences are stored in the transform domain, the watermark should be easily inserted into the transform coefficients.

This means that a watermark w can be described as a sequence w_1 to w_N , where N is the number of transform coefficients in the video sequence.

2.2.2 Inserting a Watermark

In [1], the following insertion formulas are suggested:

1. $v_i^* = v_i + \alpha w_i$
2. $v_i^* = v_i (1 + \alpha w_i)$
3. $v_i^* = v_i e^{\alpha w_i}$

Where:

- v is the video sequence to watermark, and v_i is its i :th transform coefficient.
- v^* is the output watermarked video sequence, and v_i^* is its i :th coefficient.
- w is the watermark and w_i is its i :th number.
- α is a constant.

Note that only the first of these formulas are possible to inverse for all values of v_i . The second and third cannot be inverted when $v_i = 0$.

2.2.3 Comparing a Watermark

When the owners of a video sequence v finds an illegal copy of it, v^* , they might want to find out if it contains a certain watermark. There are two basic ways for them to do that:

- Extract w_i^* from v_i^* for every i by using the inverse insertion formula and calculate the total squared difference d as:
 $d = \sum (w_i^* - w_i)^2$.

- Insert w into v to get v^{**} and calculate the total squared difference d as:

$$d = \sum (v_i^{**} - v_i^*)^2$$

If the squared difference d is lower than some threshold t , the comparison is a match. A way to generate a threshold is described in section 2.2.4.

Actually, it is not strictly necessary to use the squared difference. It would also be possible to use the cross correlation or something else that increases or decreases with the similarity.

2.2.4 Generating a Threshold

After a squared difference d has been calculated, the next thing to do is to generate the threshold t , i.e. how low d has to be in order for the watermark comparison to be considered a match.

This can be done as follows:

1. Assume that the given watermark w is not the watermark in the watermarked video sequence v^* , but some other random watermark. This means that each coefficient w_i is a stochastic variable.
2. Since the squared difference d is a sum of many stochastic variables, it is likely using roughly a normal distribution. Steps 3 to 5 are used to calculate its mean m and standard deviation σ .
3. Generate M (where M is large) random watermarks, w^1 to w^M , with the same amplitude as w .
4. Insert each w^j into v and calculate each squared difference d^j .
5. If M was chosen large enough, the sample mean of the values of d^j is roughly equal to m and the sample standard deviation is roughly equal to σ .
6. We now have that d is a stochastic variable with a normal distribution with a known mean m and standard deviation σ . This means that we can find a t such that $P(d \leq t) = e$, where e is the accepted error probability, and should be chosen to something very small.
7. The desired threshold is t . For $e = 0.01$, $t = m - 2.33\sigma$.

2.2.5 Watermarks and Predictive Coding

If inserting a watermark changes some value v_i that is used to predict some other value v_j , the distortions generated by inserting the watermark will spread to v_j .

If there is a whole sequence of predictive coded coefficients, each based on the previous one, and the watermark modifies all of them, the distortions will accumulate and the last coefficient will get all distortions, destroying it. This is obviously not unobtrusive.

There are three different ways to solve this problem:

- For each coefficient that is changed, determine all values that must be changed too to compensate for the changed prediction. This would completely solve the problem, but would make the insertion algorithm much more complicated, since coefficients are generally predicted from coefficients of previous frames, which means that there will be many values to keep track of.
- Do not insert the watermark into coefficients that will be used to predict future coefficients. Unfortunately, a vast majority of all coefficients will be used to predict coming coefficients, and those that are not can generally be completely removed without degrading the quality, which would make the watermark easily defeated.
- Do not insert the watermark into coefficients that use predictive coding themselves. This will not solve the problem completely, since a coefficient can be used to predict other coefficients even if it does not use predictive coding itself. However, at least the error generated will be constant rather than accumulating.

3 The H.264 Standard

The video standard I chose to use for this master thesis is the H.264 standard (also known as MPEG4 AVC) as it offers a significant efficiency improvement over the previous video compression standards (i.e. the video sequences get better bit rate/distortion ratios).

This chapter describes the parts of the H.264 video standard that are important to this thesis. For more information on the H.264 video standard, see [3].

3.1 Sub Blocks

A sub block is the basic part of a H.264 video stream. There are two types of sub blocks: luminance sub blocks and chroma sub blocks. Luminance sub blocks consist of 4x4 pixels whereas chroma sub blocks consist of 2x2 pixels. The blocks are transform, prediction and entropy coded.

There are several different ways how a sub block can be prediction coded, where the simplest one is that the first transform coefficient of each sub block use the first transform coefficient of neighboring sub blocks and the other transform coefficients do not use prediction coding at all.

Entropy coding can be done using two different algorithms, named CAVLC (Context Adaptive Variable Length Coding) and CABAC (Context Adaptive Binary Arithmetic Coding).

3.2 Macro Blocks

A macro block consists of 4x4 luminance sub blocks and 2x2x2 chroma sub blocks. A macro block also contains information on what kind of prediction coding its sub blocks use.

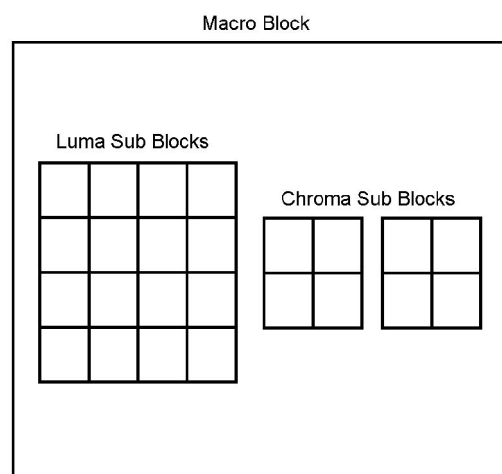


Figure 1. A macro block.

3.3 Slices and Frames

A slice contains a header and a sequence of macro blocks.

There are three types of slices: I slices, P slices, and B slices. I slices are completely intra coded, whereas P slices and B slices use predictive coding from other slices.

The slice header also contains a reference to which parameter set that is used to decode it as well as information in which order the macro blocks come in the slice.

A number of slices that together fill up the screen are called a frame.

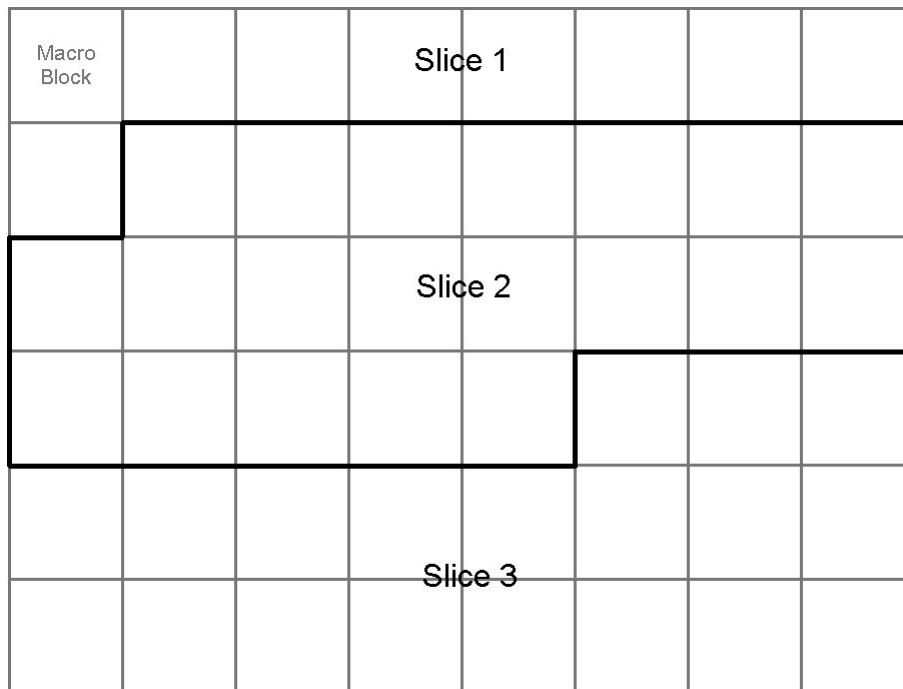


Figure 2. Example of three slices.

3.4 Parameter Sets

A parameter set contains decoding information that is used to be able to decode one or more slices. An example of information that is stored in a parameter set is the entropy coding mode flag, which describes if CABAC or CAVLC entropy arithmetic coding is used for the sub blocks.

3.5 Network Abstraction Layer Units

A network abstraction layer unit (abbreviated NAL unit) contains a header and either a slice or a parameter set.

The NAL units are placed after each other in H.264 file.

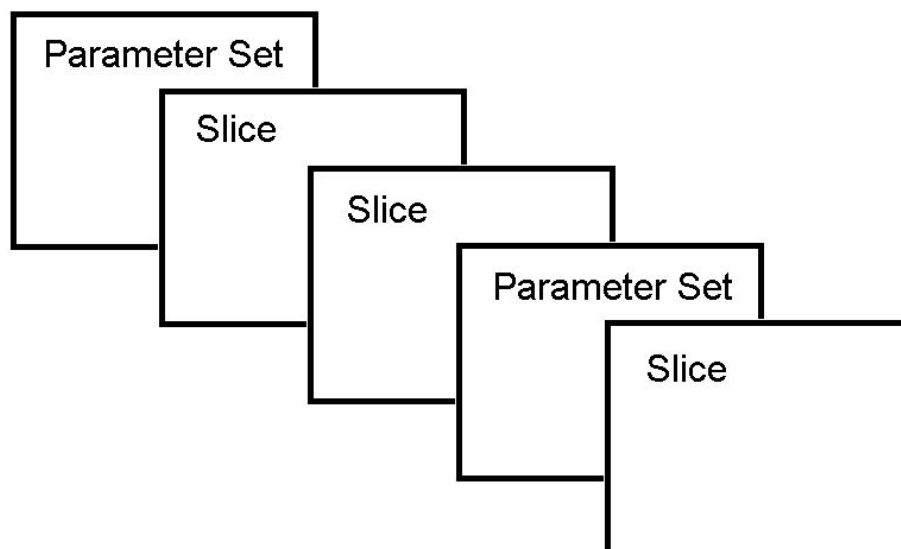


Figure 3. Example of NAL units.

4 Implementation

This chapter deals with the implementation of the program for this thesis.

The program has four main functions:

- *Generate a watermark* – This will generate a random watermark.
- *Insert a watermark* – This will insert a watermark into a video sequence and save the resulting watermarked video sequence.
- *Compare a watermark* – This will, given a watermark, a watermarked video sequence and the original non-watermarked video sequence, insert the watermark to the original video sequence and calculate the square difference (see section 2.2.3).
- *Generate a threshold* – This will generate a threshold. If the squared difference generated by a comparison is higher than the threshold, there is a match.

4.1 Generating a Watermark

To generate a watermark, use the following command line:

```
watermark generate a w
```

Where:

- a is the amplitude of the watermark.
- w is the name of the file to store the watermark in. If a file with this name already exists, it will be overwritten.

The program will generate a watermark that is 240 coefficients long. This is the same as the number of non-prediction coded luminance coefficients in a H.264 macro block that uses the simplest prediction coding scheme (see sections 3.1 and 3.2).

The coefficients in the watermark will be randomly generated with a standard deviation of a .

4.2 Inserting a Watermark

To insert a watermark, use the following command line:

```
watermark insert w v v*
```

Where:

- w is the name of the file containing the watermark you want to insert.
- v is the name of file containing the video sequence file you want to insert a watermark into.
- v^* is the name of file you want to save the watermarked video sequence to. If a file with this name already exists, it will be overwritten.

The program will do the following:

```

While v contains unread NAL units, do:
  Read the first unread NAL unit, n, from v.
  If n contains an I slice, then:
    Read the I slice, s, from n.
    Write the header of n to v*.
    Write the header of s to v*.
    While s contains unread macro blocks, do:
      Read the first unread macro block, m, from s.
      If m uses the simplest prediction, then:
        For i in 1 to 240, do:
           $m_i \leftarrow m_i \cdot (1000 + w_i) / 1000.$ 
        End for.
      End if.
      Write m to v*.
    End while.
  Else:
    Write n to v*.
  End if.
End while.

```

Where ‘ \leftarrow ’ is the assignment operator, m_i is the i :th non-predicted transform coefficient and w_i is the i :th coefficient in w .

If we compare this with the theory in section 2.2.2, we see that I use the 2:nd insertion function with $\alpha = 0.001$.

The reason why the watermark is only inserted into the macro blocks that use the simplest prediction is to prevent accumulating distortions as described in section 2.2.5.

4.3 Comparing a Watermark

To check if a video sequence contains a watermark, use the following command line:

```
watermark compare w v v*
```

Where:

- w is the name of the watermark.
- v is the name of the file containing the original, non-watermarked, video sequence, and
- v^* is the name of file containing the watermarked video sequence.

The program will do the following:

```

d ← 0.
While there are macro blocks in both v and v*, do:
  Find a macro block m in v.
  Find a macro block m* in v*.
  If m and m* uses the simplest prediction, do:
    For i in 1 to 240, do:
       $m_i \leftarrow m_i \cdot (1000 + w_i) / 1000.$ 
       $d \leftarrow d + (m_i - m_i^*)^2.$ 
    End for.
  End if.
End while.
Print d.

```

Where ‘ \leftarrow ’ is the assignment operator, m_i is the i :th non-predicted transform coefficient and m_i^* is the i :th coefficient in m^* .

4.4 Generating Threshold

To get the threshold when a watermark comparison should be considered a match, use the following command line:

```
watermark threshold w v v*
```

Where:

- w is the name of file containing a watermark. Only the amplitude of the watermark is used; the coefficients in the file are ignored.
- v is the name of the file containing the original, non-watermarked, video sequence.
- v^* is the name of file containing the watermarked video sequence.

This will do the following:

For i in 1 to 100, do:

 Generate a watermark w^* with the same amplitude as w .

 Let d_i be the result of the following command:

```
    watermark compare w* v v*.
```

End for.

Calculate the mean m and standard deviation σ of the d_i values.

Print $m - 2.33\sigma$.

Note that this algorithm is unnecessarily inefficient as it reads through all macro blocks 100 times to see if it should insert a watermark into it. If the outermost loop had been placed innermost instead, it would have been more efficient.

4.5 Watermarks Sequences

If you want a longer watermark than just one macro block, it is possible to make a watermark sequence consisting of normal watermarks.

In a Unix environment, this is done by the following command line:

```
ls w1 ... wN > s
```

In a Windows environment, it is done by the following command line:

```
dir /b w1 ... wN > s
```

In both cases:

- $w^1 \dots w^N$ is a space separated list of the watermarks to include. Wildcards (* and ?) are allowed.
- s is the name of the file to store the sequence in.

The output file s will contain references to the original watermark files, which means that if one or more of the original watermarks are removed, the watermark sequence cannot be used.

A watermark sequence can be used as a normal watermark for insertions, comparisons and threshold generations. The first watermark in the watermark sequence will be used with the first macro block in the video sequence, the second watermark with the second macro block and so forth, looping at the end of the watermark sequence.

4.6 Known Bugs and Limitations

The following limitations exist:

- The program only supports CAVLC entropy coding (see section 3.1). If the video sequence uses CABAC entropy coding, the program will exit with an error message. The program is however written to allow easy addition of CABAC support in the future.
- The program only supports slices where the macro blocks are sorted from left to right and top to bottom in the slice. Any other sorting will generate a random error message.

5 Testing

This chapter describes the testing that was performed. The video sequence used was stored in a file called video0.avc. This video sequence is 720x576 pixels large and contains 16 frames.

5.1 Basic Usage

This section deals with the basic usage of the program, i.e. generating, inserting, comparing and threshold calculation.

The test was performed as follows:

1. A thousand watermarks were generated, by executing the following command for each i in 1 to 1000:
`watermark generate 100 i.wm`
2. The first of these watermarks was inserted into video0.avc by executing:
`watermark insert 1.wm video0.avc video1.avc`
3. A threshold was calculated by executing:
`watermark threshold 1.wm video0.avc video1.avc`
Threshold: 806.549
4. The following command was executed for each i in 1 to 1000:
`watermark compare i.wm video0.avc video1.avc`

The result of the comparisons are shown in the diagram below:

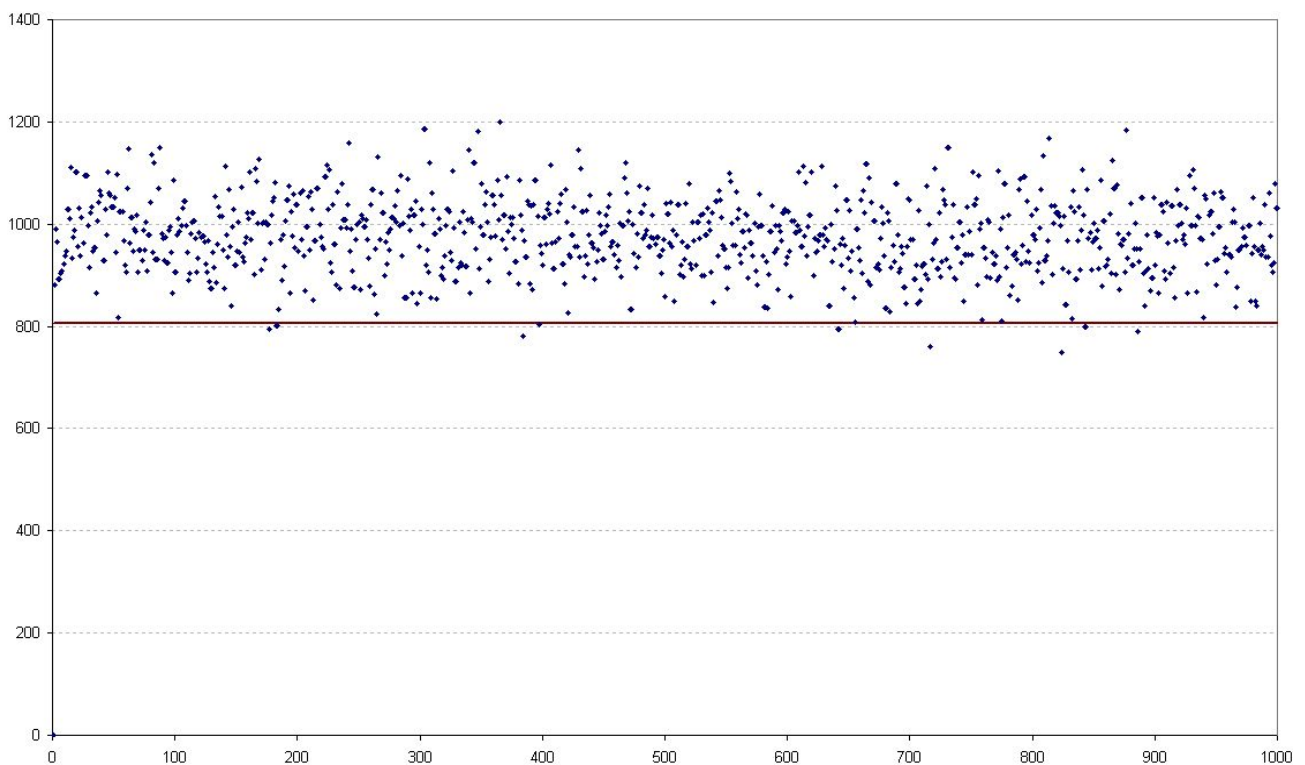


Figure 4. Diagram of results of basic insertion and comparison.

In the diagram, a dot at location (x, y) means that the x :th watermark got a squared difference of y . The line represents the threshold. As can be seen, there are only a few false positives. The first watermark, which is the one that was put into the video sequence, got a much lower squared difference than the other watermarks.

5.2 Watermarks Sequences

The program is supposed to be able to handle watermark sequences instead of normal watermarks. This section will test that it works. The following commands were executed:

```
watermark generate 100 1.wm
watermark generate 100 2.wm
dir /b 1.wm 2.wm > 12.wms
dir /b 2.wm 1.wm > 21.wms
watermark insert 12.wms video0.avc video12.avc
watermark insert 21.wms video0.avc video21.avc
watermark threshold 12.wms video0.avc video12.avc
Threshold: 750.041
watermark threshold 21.wms video0.avc video21.avc
Threshold: 807.612
watermark compare 12.wms video0.avc video12.avc
Square Difference: 0
watermark compare 12.wms video0.avc video21.avc
Square Difference: 847
watermark compare 21.wms video0.avc video12.avc
Square Difference: 847
watermark compare 21.wms video0.avc video21.avc
Square Difference: 0
```

All these results are the expected ones.

5.3 Recoding

This section deals with the testing of recoding. The test was performed as follows:

1. A thousand watermarks were generated, by executing the following command for each i in 1 to 1000:
`watermark generate 100 i .wm`
2. The first of these watermarks was inserted into video0.avc by executing:
`watermark insert 1.wm video0.avc video1.avc`
3. The video1.avc file was decoded and then coded back, using [2]. The recoded sequence was stored in video1r.avc.
5. A threshold was calculated by executing:
`watermark threshold 1.wm video0.avc video1r.avc`
Threshold: 803.884
4. The following command was executed for each i in 1 to 1000:
`watermark compare i .wm video0.avc video1r.avc`

The result of each comparison are shown in the diagram below:

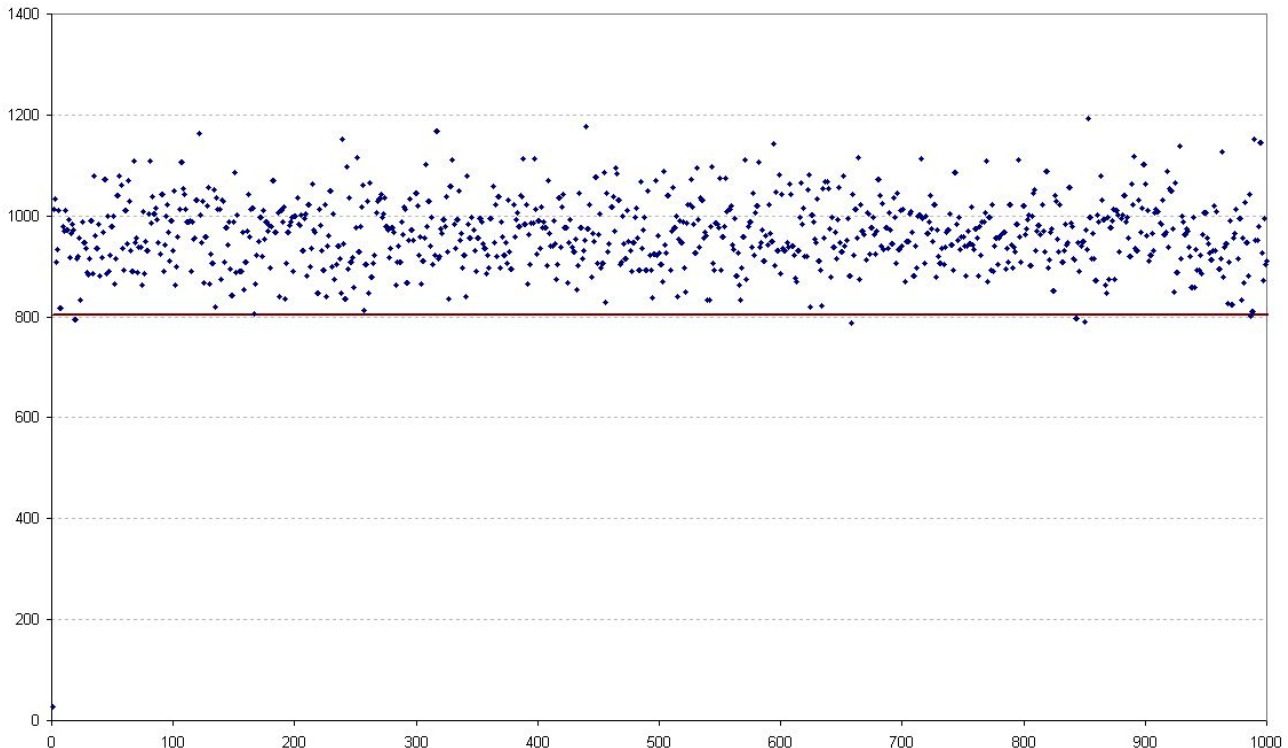


Figure 5. Diagram of results of recoding.

In the diagram, a dot at location (x, y) means that the x :th watermark got a squared difference of y . The line represent the threshold. As can be seen, there are only a few false positives. The first watermark, which is the one that was put into the video sequence, got a much lower squared difference than the other watermarks.

5.4 Double Watermark Insertion

One thing a pirate could do in an attempt to fool the watermark comparison algorithm, is to insert some random watermark in addition to the watermark already in there. The following test was made.

1. A watermark 1.wm of amplitude 100 was generated.
2. 1.wm was inserted into video sequence video0.avc to get video1.avc.
3. A couple of other watermarks, 2.wm to 6.wm of different amplitudes were generated and each added to video1.avc to get video2.avc to video5.avc.
4. The following commands was executed for each i in 2 to 6:

```
watermark compare 1.wm video0.avc videoi.avc
watermark threshold 1.wm video0.avc videoi.avc
```

The results were as follows:

Watermark	Amplitude	SqDiff	Threshold	Match
2.wm	200	620	969	Yes
3.wm	400	742	1318	Yes
4.wm	800	1674	2333	Yes
5.wm	1600	4913	5093	Yes
6.wm	3200	22495	23129	Yes

As can be seen, the pirate did not manage to defeat the watermark in this way using the amplitudes up to 3200.

5.5 Collusion

Collusion means that a group of pirates, who each have a different watermarked copy of a video sequence (i.e. fingerprinting is used), attempt to generate a video sequence whose watermark is too weak to be detected by the program.

In this thesis, two different collusion attacks are tested: collusion by averaging and collusion by swapping macro blocks.

5.5.1 Collusion by Swapping Macro Blocks

This method of collusion means that the pirates take the first macro block from the first pirate's video sequence, the next from the second pirate's sequence and so forth.

Instead inserting each watermark into the video sequence and then writing a program that would swap macro blocks, I chose to insert a watermark sequence into the video sequence, which has the same results.

In more detail, the test was as follows:

1. Generate N watermarks, where N is the number of pirates, by executing the following for each i in 1 to N :
`watermark generate 100 i.wm`
2. Make a watermark sequence, `S.wms`, that includes all N watermarks.
3. Insert `S.wms` by executing:
`watermark insert S.wms video0.avc videoS.avc`
4. Generate the threshold as follows:
`watermark threshold S.wms video0.avc videoS.avc`
5. For each i in 1 to N , execute:
`watermark compare i.wm video0.avc videoS.avc`
6. Count the number of matches in 5.

Since the test generates a different number of matches each run, it was repeated three times for each number of pirates. Here are the results:

Pirates	Matches 1	Matches 2	Matches 3	Average	Percent
3	2	2	2	2.0	67 %
6	4	4	4	4.0	67 %
12	5	6	5	5.3	44 %
25	7	7	7	7.0	28 %
50	8	13	10	10.3	21 %
100	9	9	9	9.0	9 %

As can be seen, the implementation was very robust against this type of collusion.

5.5.2 Collusion by Averaging

This method of collusion means that for each pixel in the video sequence, they choose the average of their pixel values.

Instead of writing a program that could average the video sequences, I chose to average the watermarks, which should have the same result, except that it gives different rounding errors.

In more detail, the test was as follows:

1. Generate N watermarks, where N is the number of pirates, by executing the following for each i in 1 to N :
`watermark generate 100 i.wm`
2. Calculate the average watermark, `A.wm`.
3. Insert `A.wm` by executings:
`watermark insert A.wm video0.avc videoA.avc`
4. Generate the threshold as follows:
`watermark threshold A.wm video0.avc videoA.avc`
5. For each i in 1 to N , execute:
`watermark compare i.wm video0.avc videoA.avc`
6. Count the number of matches in 5.

Since the test generates a different number of matches each run, it was repeated three times for each number of pirates. Here are the results:

Pirates	Matches 1	Matches 2	Matches 3	Average	Percent
3	3	3	3	3.0	100 %
6	6	5	5	5.3	89 %
12	7	9	4	6.7	56 %
25	3	0	1	1.3	5 %
50	9	0	6	5.0	10 %
100	1	0	8	3.0	3 %

As can be seen, the implementation was less robust against this kind of collusion than the collusion in 5.5.1, but still quite robust.

5.6 Unobtrusiveness

In this section, the unobtrusiveness of the watermarks is tested. Watermarks of different amplitudes were inserted into the video sequence and the signal to noise ratio (SNR), in decibel, of the worst frame was calculated.

Amplitude	SNR
100	43
200	44
400	44
800	43
1600	39
3200	34

Below are shown the worst frame of the video sequences watermarked with amplitudes of 800, 1600 and 3200 respectively. The corresponding frame of the original non-watermarked video sequence is provided for comparison:



Figure 6. Non-watermarked



Figure 7. Amplitude 800



Figure 8. Amplitude 1600



Figure 9. Amplitude 3200

By looking closely, it can be seen that all of them have visible differences. However, the differences are only severe for the watermark with amplitude 3200.

5.7 Time Consumption

The following commands were executed and the time was measured:

```
watermark generate 100 1.wm
```

Time: 0.011 seconds.

```
watermark insert 1.wm video0.avc video1.avc
```

Time: 0.526 seconds.

```
watermark compare 1.wm video0.avc video1.avc
```

Time: 0.909 seconds.

```
watermark threshold 1.wm video0.avc video1.avc
```

Time: 88 seconds.

As a comparison, decoding the video sequence takes 35 seconds.

Note that for very long video sequences, these execution times might be limiting. To speed up the program, it is would be possible to only insert watermarks into every N:th I slice, where N is some appropriate number. The threshold calculation algorithm can be improved as described in 4.4.

The testing computer used had the following hardware and software configuration:

- An Athlon XP 2000+ processor.
- 512 MB PC2100 memory.
- A Seagate Barracuda IV 7200 rpm hard disk with 8 MB cache.
- The Windows 2000 operating system.

6 Conclusion

A watermarking technique for H.264 video sequences has been developed. The hidden information is inserted by altering the transform coefficients.

In chapter 5, the following image manipulations and attacks were performed on watermarked video sequences:

- Recoding
- Double watermark insertion
- Collusion

The program passed all these tests without any problem, despite using very small watermark (an amplitude of 100, which was used for these tests, means that the transform coefficients are only increased or decreased by 10 % on average).

Since these tests are representative for most image manipulations and attacks, this shows that the watermarks generated by the program can be expected to be rather robust, at least for the video sequence used.

By looking at the watermarked video sequences and comparing them to the originals, or measuring the signal to noise ratio, one can also see that the watermarks are unobtrusive.

The times were also measured. Compared to coding and decoding a H.264 video stream, the time it takes to insert and extract watermarks was much less. Calculating a threshold takes roughly double the time as decoding the sequence, though.

7 References

- [1] Ingemar J. Cox, Joe Kilian, F. Thomson Leighton, and Talal Shamooh, “Secure Spread Spectrum Watermarking for Multimedia”, 1997
- [2] Karsten Sühling, “JM Software”, <http://bs.hhi.de/~suehring/tml/>
- [3] Thomas Wiegand, Gary Sullivan, ”Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification”, ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC, 2003

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: <http://www.ep.liu.se/>

© David Bergkvist