TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Rechnernetze

Masterarbeit

AUTOMATISIERTES ENDE-ZU-ENDE-VIDEO-VERIFIKATIONSSYSTEM FÜR VERTEILTE VIDEO-KONFERENZSYSTEME

Christian Rose
Mat.-Nr.: 3910609

Betreut durch:
Dr.-Ing. Tenshi Hara

und:
Dr.-Ing. Thomas Springer

Verantwortlicher Hochschullehrer:
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Externer Betreuer (LogMeIn):
Dipl.-Inf. (FH) Benjamin Prestele

Eingereicht am 22. Januar 2018 (ursprünglich: 05. Januar 2018)

ERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit selbständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Dresden, 22. Januar 2018 (ursprünglich: 05. Januar 2018)

ZUSAMMENFASSUNG

Test Driven Development ist eine verbreitete Methode in der professionellen Softwareentwicklung großer Systeme, bei denen die einzelnen Module, aus denen diese Systeme bestehen, aufgrund von Abhängigkeiten und wechselseitiger Einflussnahme nicht in einem Vakuum betrachtet werden können. Deshalb werden häufig Integrationstests eingesetzt, bei denen mehrere Module zusammen als Einheit getestet werden. Bei verteilten Multimedia-Bearbeitungssystemen gestaltet sich aufgrund ihrer Komplexität ein solcher Ende-zu-Ende-Test oft als schwierig. In derartigen Multimedia-Bearbeitungssystemen sind dabei häufig gleich eine Vielzahl von unterschiedlichen Modulen und Techniken verarbeitet, die an der verlustlosen und verlustbehafteten Kompression beteiligt sind. Änderungen an einem Modul führen im Verlauf der Entwicklung und Wartung eines solchen Systems zu sich stark voneinander unterscheidenden Ausgabedaten. Aufgrund einer fehlenden klaren Definition der Ausgabedaten ist eine automatische Unterscheidung zwischen ungewollten Fehlern und gewollten Kompressionsartefakten schwer, sodass die traditionellen Software-Testmethoden an ihre Grenzen stoßen. In dieser Arbeit wird eine Methodik vorgestellt, die mithilfe eines semi-fragilen Wasserzeichens einen Ende-zu-Ende-Test eines Video-Konferenzsystems ermöglicht. Die Integrität des Wasserzeichens wird dabei als Fehlerindikator verwendet. Durch den Einsatz eines fehlerkorrigierenden Codes lassen sich die gefundenen Fehler im Bild lokalisieren. Durch das zusätzliche Einbetten von zeitlichen Informationen können zudem Aussagen über die zeitliche Integrität getroffen werden. Die vorgestellte Technik ist so in der Lage, unterschiedliche zeitliche Fehler, sich wiederholende Fehler im Bild und Übertragungsfehler zu erkennen.

INHALTSVERZEICHNIS

1 EINFÜHRUNG

EINLEITUNG

„Zu 100 Prozent fehlerfreie Software gibt es nicht."

Claus Heinrich,
ehemaliger SAP-Vorstand

1.1 MOTIVATION

Das Entwickeln von Software ist ein komplexes Themengebiet. Moderne Softwaresysteme bestehen dabei häufig aus mehreren Millionen Zeilen Quellcode [31, 32, 33]. Diese Millionen Zeilen werden meist von großen Teams, die teilweise geografisch getrennt voneinander arbeiten, entwickelt. Aufgrund der Komplexität, des Koordinationsaufwands und der Größe dieser Softwaresysteme sind Fehler nicht ganz auszuschließen. Einige Entwickler und Manager halten die Entwicklung von fehlerfreier Software generell für nicht realisierbar. So hat der ehemalige SAP-Vorstand Claus Heinrich gesagt: „Zu 100 Prozent fehlerfreie Software gibt es nicht." [1].

Bei verteilten Systemen erhöht sich das Fehlerpotential noch einmal. Sie bestehen häufig aus mehreren kooperierenden Codefragmenten, bei denen es nicht nur darauf ankommt, dass kein Fehler in den einzelnen Komponenten existiert, sondern auch darauf, dass das Zusammenspiel der Komponenten funktioniert. Dabei können Fehler in einzelnen Komponenten Einfluss auf andere Komponenten im System haben und so eine Art Kettenreaktion auslösen, die das gesamte System beeinträchtigt. Das Minimieren von Fehlern in den Softwaresystemen ist daher eine große Herausforderung der modernen Softwareentwicklung.

Die Methodik des „Test Driven Development (TDD)" hat sich als probates Mittel erwiesen, Fehler bereits früh während der Softwareentwicklung zu erkennen und ihre Anzahl so zu begrenzen. Dabei werden beim TDD anhand der Spezifikationen zuallererst Tests geschrieben und die Software danach anhand dieser Testfälle entwickelt [6, p. XIX]. Um auch das Zusammenspiel der einzelnen Softwarekomponenten sicherzustellen, kommen hierbei Integrationstests zum Einsatz. Bei Integrationstests werden mehrere oder sogar alle Komponenten eines Systems von Anfang bis Ende als Einheit getestet. Solch ein automatisierter Integrationstest ist einfach zu realisieren, wenn für gegebene Eingabedaten eine Ausgabe festgelegt ist. In einem solchen Fall können die Ausgabedaten einfach mit den erwarteten Werten abgeglichen werden.

Bei Software zur Verarbeitung und Kompression von Multimediadaten gestaltet sich genau dieses Definieren von Ausgabewerten häufig schwierig. Bei Videoverarbeitungssystemen werden die Ausgabedaten nicht auf mathematisch spezifizierte Werte hin optimiert, sondern auf die menschliche Wahrnehmung. Die Ausgaben, die durch ein solches System entstehen, können sich dabei stark voneinander unterscheiden, vom Betrachter allerdings als qualitativ gleichwertig angesehen werden. Eine Änderung an einer Komponente kann also die ausgegebenen Pixelwerte stark verändern und trotzdem ein gültiges Bild erzeugen. Bei Videokompressionsverfahren wie beispielsweise beim H.264-Standard gibt es eine Vielzahl an hintereinander geschalteten, verlustlosen und verlustbehaften Kompressionsverfahren. Durch das Zusammenspiel der unterschiedlichen Komponenten ist dabei nicht immer einfach voraussagbar, welchen Einfluss eine Änderung an einer einzelnen Komponente auf die darauffolgenden haben wird. Eine Veränderung an einzelnen Komponenten, wie sie bei der (Weiter)Entwicklung von Software häufig vorkommt, kann so erheblichen Einfluss auf die Ausgabedaten haben. Bei Änderungen an der Software können auch Fehler in die Software

eingebracht werden. Die Auswirkungen der Fehler sind dabei vielfältig, viele werden sich aber in irgendeiner Form auf das ausgehende Video auswirken.
Eine automatisierte Unterscheidung der gewollten - für einen Betrachter nicht wahrnehmbaren - Veränderungen von den durch Fehler entstehenden Veränderungen ist schwer. Um die genauen Auswirkungen einzelner Veränderungen zu testen, wäre eine Möglichkeit, vor Änderungen an der Software zeitaufwendige und komplexe Berechnungen durchzuführen, um den erwarteten Wert der Veränderung zu ermitteln. Diese Berechnungen müssten dabei auch den Einfluss der unterschiedlichen hintereinandergeschalteten Kompressionsverfahren in Betracht ziehen. Alternativ müsste nach einer Änderung an einem der Kompressionsmodule eine zeitaufwendige händische Verifikation der Daten getätigt werden.

In dieser Arbeit wird ein Konzept erarbeitet, welches einen automatisierten Ende-zu-Ende-Test eines Videoverarbeitungssystems erlaubt. Die konkrete Umsetzung dieses Testsystems wird dabei mit dem Fokus auf ein verteiltes Videokonferenzsystem untersucht. Dabei wird zunächst ein Konzept erarbeitet, welches anschließend prototypisch umgesetzt wird. Dabei wird es zu Testzwecken während der Entwicklung betrieben und soll nicht im Produktivbetrieb eingesetzt werden. Diese Arbeit ist in Kooperation mit der LogMeIn Inc. (GetGo Germany GmbH) entstanden.

1.2 STRUKTUR

Im Anschluss an dieses Einführungskapitel werden in Kapitel 2 einige Grundlagen erklärt, die für die folgenden Kapitel wichtig sind. In Kapitel 3 werden anschließend einige bestehende Forschungsansätze im Bereich der Video-Verifikation und in verwandten Themenbereichen untersucht. Das darauffolgende Kapitel 4 beschreibt die Anforderungen die vom Partner, der LogMeIn Inc., vorgegeben wurden. Aufbauend auf dem vorhergehenden Kapitel wird in Kapitel 5 eine Anforderungsanalyse vorgenommen. In Kapitel 6 wird, basierend auf den verwandten Arbeiten und anhand der in der Anforderungsanalyse herausgearbeiteten Anforderungen, ein Konzept entwickelt. Kapitel 7 beinhaltet eine grobe Umschreibung der Implementierung, wobei auf einige Herausforderungen eingegangen wird, die sich bei der Umsetzung des Konzeptes ergeben haben. In Kapitel 8 wird der Einfluss einiger Parameter auf das Konzept untersucht und auf die in der Anforderungsanalyse herausgearbeiteten Anforderungen hin optimiert. Kapitel 9 stellt anschließend die Ergebnisse dar und wertet aus, inwiefern die Anforderungen erfüllt werden konnten. Kapitel 10 gibt eine finale Zusammenfassung der Ergebnisse. Dabei wird ein Fazit gezogen und ein Ausblick auf weitere Forschungsansätze gegeben.

2 GRUNDLAGEN

GRUNDLAGEN

In diesem Kapitel werden einige grundlegende Konzepte und Techniken beschrieben, die im Rahmen dieser Arbeit wichtig werden. Dabei wird sich die Beschreibung, wenn nötig, auf die für diese Arbeit wichtigen Aspekte beschränken. Die in diesem Kapitel vorgestellten Grundlagen erheben nicht den Anspruch, eine vollumfängliche Zusammenfassung der Thematiken zu sein. Für interessierte Leser werden Verweise auf weiterführende Informationen angeboten.

2.1 ENCODER - DECODER

Unter einem Encoder wird im Rahmen dieser Arbeit ein Kompressionstool für Videos verstanden. Das bedeutet ein Werkzeug, welches durch Reduzieren von redundanten Informationen, Umcodieren der Videodaten und visuell unauffällige Veränderungen der Videodaten eine geringere Bitrate des Videos erreicht. Der Decoder ist dabei das Tool, welches die encodierten Daten wieder in eine Form umcodiert, die sich anzeigen lässt. Dabei wird im Rahmen dieser Arbeit die verlustbehaftete Kompression ausdrücklich eingeschlossen. Das bedeutet, dass die decodierten Videodaten nicht exakt den encodierten Daten entsprechen müssen, weil durch das Vorgehen des Encoder-Decoder-Paars bestimmte Informationen unwiderruflich verloren gehen können.

2.2 DISKRETE KOSINUSTRANSFORMATION

Die Diskrete Kosinustransformation, kurz DCT (aus dem Englischen „discrete cosine transformation"), ist eine Technik, mit der sich Signale umkehrbar transformieren lassen [4, p. 108]. Das Signal wird dabei so transformiert, dass es als eine Reihe von Kosinusfunktionen (häufig als Basisfunktionen bezeichnet) darstellbar ist[4, p. 112]. Die DCT ist dabei eine Transformation, die in vielen verlustbehafteten Bild- und Videokompressionsverfahren Anwendung findet[4, p. 112]. Dabei ist die DCT von der diskreten Fourier-Transformation abgeleitet[4, p. 112]. Es sind DCT Typen von 1 bis 4 bekannt, für die Datenkompression kommt in der Regel die DCT vom Typ 2 zum Einsatz[4, p. 112]. Matrizen, die durch die DCT transformiert werden, werden im Rahmen dieser Arbeit als DCT-Koeffizienten bezeichnet. Dabei wird dem oberen linken Wert der so entstandenen Matrix eine besondere Bedeutung zugemessen, da er selbst keine Struktur enthält, sondern die Helligkeit des Bildes steuert. Dieser wird folgend als „DC-Wert" bezeichnet. Im Rahmen dieser Arbeit wird zur Berechnung die DCT-Implementierung des Typs 2 der SciPy-Bibliothek[29] verwendet. Die Funktion ist in Gleichung 2.1 dargestellt.[1] Dabei ist N eine Dimension der Blockgröße, k die Position des Pixelwertes, der gerade bearbeitet wird und x ist der Pixelwert.

$$y(k) = 2 \cdot \sum_{n=0}^{N-1} x[n] \cdot cos(\pi \cdot k \cdot \frac{2n+1}{2 \cdot N}), 0 <= k < N. \tag{2.1}$$

Die Basisfunktionen, die durch Formel 2.1 für einen 8×8 Pixel großen Block entstehen, sind in Abbildung 2.1 illustriert.

[1] Zur Vereinfachung ist hier lediglich die eindimensionale DCT Funktion dargestellt. Bei einer zweidimensionalen DCT, wie sie bei der Verarbeitung von Bildern zum Einsatz kommt, werden die Daten zweimal DCT-transformiert, einmal in horizontaler und einmal in vertikaler Richtung.

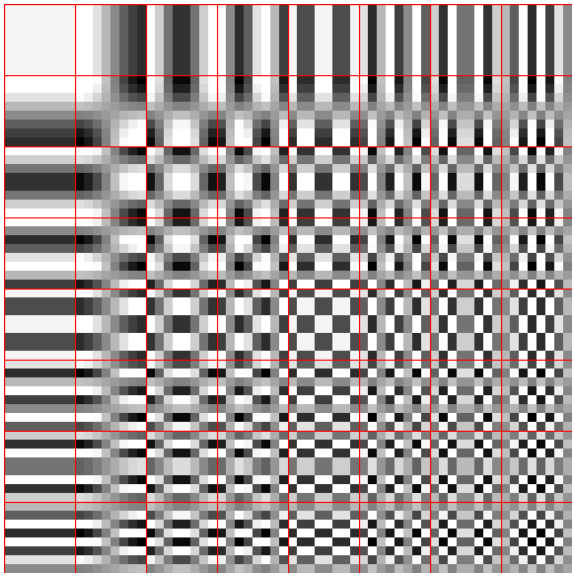


Abbildung 2.1: Zweidimensionale Darstellung der DCT-Frequenzen für einen 8×8 großen Block.

2.3 H.264

In diesem Abschnitt wird ein kurzer und oberflächlicher Überblick über den H.264/AVC-Standard gegeben. Dabei findet in dieser Arbeit keine komplette Besprechung des H.264/AVCs statt, da dies zum einen den Rahmen dieser Arbeit sprengen würde und zum anderen der Kompressionsstandard nicht Fokus dieser Arbeit ist. Hier werden also vor allem Verfahren behandelt, die für die folgende Arbeit von Bedeutung sind. Für weitere Informationen zum Thema H.264/AVC-Standard bietet sich das Standardwerk von Iain E. Richardson [3] an.

Der H.264/AVC ist ein Video-Kompressionsstandard [3, p. 81], der aus mehreren unterschiedlichen Werkzeugen zusammengesetzt ist [3, p. 83]. Die Bilder des Videos werden beim Encodieren in sogenannte Makroblöcke aufgeteilt, dies sind $N \times M \mid N,M \in \mathbb{N}$ Pixel große Regionen[3, p. 31]. Grundlegend lässt sich der Ablauf des Encoders in folgende Schritte aufteilen[3, p. 81]:

- prediction
- transform
- entropy encode
- entropy decode
- inverse transform
- reconstruction

Die „Prediction“ kann weiter in Inter- und Intra-Frame-Prediction unterteilt werden [3, p. 85]. Intra-Frame-Prediction greift dabei lediglich auf die Information des momentanen Bildes zu, wohingegen die Inter-Frame-prediction auch auf zeitlich angrenzende Bilder zugreift [3, p. 85]. Bei der Inter-Frame-Prediction wird zwischen drei Makroblock-Typen unterschieden: **I**,**P**,**B** [3, p. 101]. Makroblöcke des Typs **I** enthalten alle nötigen Informationen, die zu ihrer Decodierung und Darstellung benötigt werden [3, p. 101]. Makroblöcke des Typs **P** und **B** speichern lediglich

Differenzen zum vorhergehenden und ggf. nachfolgenden Block und brauchen weitere Blöcke, um korrekt angezeigt zu werden[3, p. 101]. Im Rahmen dieser Arbeit werden Bilder, die jeweils nur aus I-,P-,B-Makroblöcken bestehen, mit den entsprechenden Präfixen bezeichnet (I-Bild bzw. Key-Bilder, P-Bild und B-Bild). Bei der Inter-Frame-Prediction kommen dabei Blockgrößen von 4×4 Pixel bis zu 16×16 Pixel zum Einsatz.

Bei der Transformation werden Bilddaten in Blöcke aufgespalten und mit einer angenäherten Diskreten Kosinustransformation (DCT) transformiert [3, p. 87]. Die so entstandenen Blöcke werden anschließend quantisiert, was bedeutet, dass die DCT-Koeffizienten durch einen Ganzzahl-Wert dividiert werden [3, p. 87]. Dies kann entweder mit festen Werten oder mit festgelegten Tabellen[3, p. 204] geschehen. Beim „Entropy Encoding“ kommt „Variable Length Coding“ zum Einsatz [3, p. 89].

2.4 BCH-CODE

Der BCH-Code, benannt nach seinen Erfindern Bose, Chaudhuri und Hoquenghem [2, p. 162] ist ein fehlerkorrigierender Code [2, p. 162]: er kann Änderungen im Bitstrom nicht nur detektieren, sondern Fehler anhand redundant gespeicherter Informationen auch korrigieren. Als Vertreter sogenannter zyklischer Codes können Fehler im Bitstrom jedoch nur in einem gewissen Umfang detektiert werden: werden zu viele Bits verändert, erscheint das empfangene Codewort wieder als gültig. Der BCH-Code gehört zur Klasse linearer Blockcodes, bei denen die Transformation vom Quellcodewort zum Kanalcodewort eine Gruppe algebraischer Strukturen definiert [2, p. 162]. Dabei sind die Koeffizienten der Codepolynome als Elemente über GF(2)[2, p. 162] definiert. Es wird im Rahmen dieser Arbeit zwischen Quellcodewort und Kanalcodewort unterschieden. Das Quellcodewort beschreibt dabei eine unveränderte Zeichenfolge, zu der mithilfe des BCH-Codes Redundanzen hinzugefügt werden. Die so entstandene Zeichenfolge aus Quellcodewort und Redundanzen wird als Kanalcodewort bezeichnet.

2.5 BEGRIFFLICHKEITEN

Im Rahmen dieser Arbeit werden einige Begriffe verwendet, über die nachfolgend eine Übersicht gegeben wird:

Videoverarbeitungskette: Dies bezeichnet eine Kette von hintereinander ausgeführten Videobearbeitungsschritten, meistens ein oder mehrere Encoder-Decoder-Paare.

Eingangsdaten: Dies beschreibt die unveränderten Originaldaten, die in eine solche Videoverarbeitungskette hineingehen.

Ausgabedaten: Dies beschreibt analog zu den *Eingangsdaten* die durch eine Videoverarbeitungskette veränderten Daten.

Trägermedium: Dies beschreibt die originalen unveränderten Daten, die dazu verwendet werden, ein Wasserzeichen einzubetten.

Informationsträger: Eine einzelne Einheit des *Trägermediums*. Im Fall von Bildern entspricht dies einem einzelnen Pixel.

Block: Als Block wird nachfolgend ein rechteckiger Bereich eines einzelnen Bildes bezeichnet.

Blockgröße: Die Blockgröße ist definiert als Breite $\times$ Höhe eines Blockes.

3 VERWANDTE ARBEITEN

VERWANDTE ARBEITEN

Im folgenden Abschnitt wird herausgearbeitet, welche Möglichkeiten und Techniken existieren, um eine Videoverarbeitungskette, bestehend aus Encoder und Decoder, automatisiert zu testen. Dabei werden zusätzlich zu den Techniken, die für Videoverarbeitungsketten verwendet werden, auch solche Techniken untersucht, die auf Einzelbildern mit Einzelbildkompression arbeiten. Diese sind interessant, da die verlustbehaftete Videokompression[3, p. 179] und Einzelbildkompression[5, p. 539] bei der verlustbehafteten Kompression teilweise mit ähnlichen Techniken arbeiten.

3.1 QUALITÄTSMETRIKEN

Wenn es um die Fehlerfreiheit von Video- und Bilddaten geht, stößt man häufig auf den Themenbereich der Qualitätsmetriken. Dabei sind Qualitätsmetriken nicht primär dazu geschaffen, Fehler im Bild zu finden, sondern die Qualität von Bildern oder Videos einzuschätzen bzw. die Qualitätseinbußen eines Bildes anhand einer Referenz zu ermitteln. Eine sehr populäre Metrik ist die peak signal-to-noise ratio(PSNR) [3, p. 20]. Diese Metrik errechnet den „Mean squared error (MSE)“ zwischen zwei Bildern und bildet ihn auf einer logarithmischen Skala ab [3, p. 20].

Berechnet wird der PSNR nach folgender Formel:

$$\mathbf{PSNR}_{dB} = 10log_{10} * \frac{(2^n - 1)^2}{\mathbf{MSE}}$$

Allgemein können Qualitätsmetriken in drei Kategorien unterteilt werden, wie von Süsstrunk et al. beschrieben: Full-Reference, No-Reference und Reduced-Reference [21]. Der PSNR fällt hierbei unter Full-Reference, die Qualität eines gegebenen Bildes wird also anhand des unveränderten Referenzbilds untersucht. No-Reference bedeutet dementsprechend, dass keine Referenzdaten zur Verfügung stehen. Diese Qualitätsmetriken bilden dabei Teile der menschlichen Wahrnehmung nach und geben darauf aufbauend eine Qualitätseinschätzung ab. Reduced-Reference stellt hierbei eine Kombination von beiden dar, wobei nur Teilinformationen eines Referenzbildes zur Verfügung stehen. Qualitätsmetriken sind nicht primär für die Erkennung von Fehlern zuständig, sondern bemessen die Qualität eines Mediums.

3.2 FEHLERERKENNUNG

Zum direkten Testen einer Blackbox gibt es beispielsweise den Ansatz von Superiori et al. In diesem Ansatz schlagen Superiori et al. vor, die Fehlererkennung durch eine Syntaxanalyse auf Decoderseite mit der Analyse eines Differenzbildes zu kombinieren [22]. So beziehen sie die zeitliche Dimension mit ein und errechnen den Unterschied zweier aufeinanderfolgender Bilder [22]. Dabei wird davon ausgegangen, dass ein Non-IDR Bild F und das darauffolgende Bild F_{+1} sich nicht stark voneinander unterscheiden, vorausgesetzt F_{+1} ist kein neues I-Bild. Anhand einer pixelweisen Differenzbildung, d.h. $D = |F - F_{+1}|$, wird die Differenz berechnet. Fehler in der Verarbeitung würden sich nach der Annahme von Superiori et al. in blockartigen Artefakten ausbilden, die sich leicht durch eine Kantendetektion erkennen lassen[22]. Superiori et al. schlagen zusätzlich noch den Einsatz einer Syntaxanalyse vor, bei der während des Decodierens untersucht wird, ob die erhaltenen Daten syntaktisch valide sind [22].

3.3 INTEGRITÄTSPRÜFUNG

Als weitere Möglichkeit, Bilddaten auf Fehler zu überprüfen, wäre die Verwendung von künstlichen Bildinhalten mit Integritätsprüfung denkbar. Unter Integritätsprüfung wird im Rahmen dieser Arbeit eine Prüfung verstanden, die es erlaubt, die Validität der Daten ohne Zusatzinformationen zu prüfen. Eine solche Integritätsprüfung ist zum Beispiel durch einen QR-Code realisierbar. In einem QR-Code ist ein fehlerkorrigierender Code mit eingebaut, der es erlaubt, fehlerhaft oder nicht erkannte Bereiche ohne zusätzliche Informationen wiederherzustellen [27]. Dabei kann der verwendete eingebettete Code auch zur Lokalisierung dieser Bereiche verwendet werden [2, p. 198].

1994 wurde von der Firma Denso Wave der QR-Code vorgestellt [27]. Der QR-Code gehört dabei zu den zweidimensionalen Barcodes [8] und wurde extra mit dem Ziel erstellt, maschinell lesbar zu sein[27]. Durch den Einsatz von fehlerkorrigierenden Codes soll ein zuverlässiges Lesen garantiert werden, sodass der Inhalt auch nach Beschädigung oder Verschmutzung noch extrahiert werden kann [27]. Dabei kann eine Fehlerkorrektur von bis zu 30% der Bits erreicht werden. Durch diese Fehlererkennungseigenschaften und „Verzerrungskompensation"[27] sind QR-Codes mit dem Gedanken entwickelt worden, auch unter schlechten Bedingungen und mit bestimmten Veränderungen am QR-Code lesbar zu sein.

3.4 WASSERZEICHEN

Die Forschung an digitalen Wasserzeichen (nachfolgend Wasserzeichen genannt) hat gezeigt, dass sie dazu eingesetzt werden können, Fehler [20] oder gewollte Manipulationen [13] an Mediendaten zu erkennen.

In der Forschung ist eine Vielzahl von unterschiedlichen Ansätzen zur Einbettung von Wasserzeichen bekannt, die für verschiedene Einsatzzwecke entwickelt wurden. Im folgenden Abschnitt soll ein Überblick über einige solche Wasserzeichen gegeben werden, die zur Erkennung von Fehlern bzw. Manipulationen an Bild- und Videodaten verwendet werden können.

Welche Anwendung durch ein Wasserzeichen erreicht werden kann, hängt teilweise bereits von der Einbettungstechnik ab. Hierbei gibt es unterschiedliche Ansätze: So setzen einige Techniken auf den Einsatz im Encoder und Decoder, sodass sie direkt in die Arbeit des Encoders eingreifen bzw. auf den bereits encodierten Daten arbeiten [10, 11, 14, 20, 13]. So können Wasserzeichen, die im Encoder arbeiten, zum Beispiel durch die Manipulation von „Motion-Vektoren" eingebettet werden[10, 11]. Andere Techniken nutzen die „Inter-Frame-Prediction" [14] des Encoders, um ihre Informationen zu verstecken.

Einige Techniken arbeiten direkt auf den Daten des Bildes oder auf einer transformierten Form [16, 15, 17, 19, 12].

Da in einigen verlustbehafteten Kompressionsverfahren ein großer Teil der Kompression in den durch die DCT-transformierten Koeffizienten geschieht [3, p. 179], kann durch die Verwendung der DCT-Koeffizienten eine bessere Voraussage getroffen werden, welche Veränderungen durch die Kompression stattfinden [18] und das Wasserzeichen entsprechend angepasst eingebettet werden.

Weitere Ansätze setzen darauf, die Informationen in den Teilen des Bildes zu verstecken, welche wenig sichtbar sind. So setzen Pröfrock et al. darauf, ihre Informationen in Kanten im Bild zu verstecken. Die Annahme dabei ist, dass Kanten innerhalb des Bildes meist zentraler Bestandteil des Bildes sind und Encoder diese seltener verändern [15].

Eine Möglichkeit ein Wasserzeichen zu realisieren, ist das Spreading [7, p. 279ff] (In dieser Arbeit frei als Streuen übersetzt). Dabei werden die Veränderungen am Wasserzeichen nicht punktuell vollzogen, sondern über das Trägermedium gestreut, sodass die Veränderungen über das Trägermedium verteilt werden [7, p. 279ff]. In diesem konkreten Beispiel wird ein Signal so verändert, dass sich das Integral des Signals verschiebt. Beim Extrahieren wird untersucht, ob das Integral des Signals oberhalb bzw. unterhalb eines Schwellwertes liegt. Abhängig von der Lage des Integrals wird das Wasserzeichen so als 1 oder 0 codiert. Durch die Streuung kann die Veränderung von einzelnen Werten ausgeglichen werden [7, p. 279ff].

3.4.1 Allgemeines

Abhängig davon, welches Ziel durch das Wasserzeichen erreicht werden soll, unterscheiden sich die Wasserzeichen signifikant. Dabei können Wasserzeichen in drei Kategorien aufgeteilt werden: fragil, robust und semi-fragil[18].

Fragil: Bei fragilen Wasserzeichen handelt es sich um Wasserzeichen, die so entwickelt werden, dass sie durch Veränderung am Trägermedium leicht kaputtgehen. Sie werden häufig angewendet, um Manipulationen am Trägermedium zu erkennen. Dabei ist das Ziel, jegliche Manipulation inklusive Kompression und Bildverarbeitung zu erkennen. Sie werden daher so entwickelt, dass sie bereits durch kleinste Veränderungen an den Bild- oder Videodaten kaputtgehen. Dies kann zum einen dazu verwendet werden, jegliche Veränderungen des Bildes zu erkennen, was auch Kompression einschließt. Solche Wasserzeichen können auch auf bereits komprimierte Daten angewendet werden, um zu verhindern, dass sie nach der Kompression noch weiter unerkannt verändert werden. Auch denkbar ist der Einsatz als Authentizitätsmerkmal, um festzustellen, ob es sich bei dem vorhandenen Bild um das Original handelt.

Robust: Robuste Wasserzeichen haben genau die entgegengesetzten Entwurfsziele. Diese Wasserzeichen werden so entwickelt, dass sie möglichst jeglicher Manipulation widerstehen, also auch bei starken Veränderungen des Bildinhalts weiterhin unbeschadet extrahiert werden können. Dies ist beispielsweise für den Urheberschutz wichtig. So soll bei urheberrechtlich geschützten Werken auch nach gezielter Manipulation weiterhin der Urheber zu erkennen sein.

Semi-fragil: Semi-fragile Wasserzeichen kombinieren beide oben genannten Eigenschaften, wobei sie robust gegen gewisse Manipulationen sein sollen, jedoch fragil gegen andere. Häufig werden diese „hybriden" Ansätze dazu verwendet, Manipulationen von Video- oder Bildmaterial zu erkennen, auch nachdem diese durch verlustbehaftete Kompression verändert wurden [19, 18].

3.4.2 Wasserzeichen zur Fehlererkennung

In der Forschung gibt es bereits einige Ansätze, Wasserzeichen zur Erkennung verschiedener Fehler in Video- und Bildbearbeitungssoftware anzuwenden.

Einige Arbeiten beziehen sich auf Ansätze, die dazu verwendet werden, Fehler auf Netzwerkebene zu überprüfen, dabei werden die Wasserzeichen auf die bereits komprimierten Werte angewendet. Durch den Einsatz nach der Kompression werden hier fragile Wasserzeichen angewendet, da jegliche gewollte Veränderung bereits durch die Kompression vollzogen wurde [20].

Dies wiederum erlaubt den Einsatz von relativ einfachen Techniken, um das Wasserzeichen einzubetten sowie auszulesen und ermöglicht häufig eine relativ hohe Einbettungskapazität. Minghua Chen et al. beispielsweise verändern die bereits durch die DCT transformierten und durch den Encoder quantisierten Werte so, dass anschließend alle Werte gerade sind[20]. Nach der Übertragung und vor dem Decoder kann überprüft werden, ob die ankommenden Daten alle geraden Werten entsprechen. Sollte dies nicht der Fall sein, kann man sicher sein, dass die Daten während der Übertragung verändert wurden.

David L. Robie und Russel M. Merserau schlagen eine ähnliche Technik vor. Genau wie schon Minghua Chen et al. verändern sie die Daten, nachdem sie DCT-transformiert und quantisiert wurden[13]. Zusätzlich schlagen sie jedoch vor, eine binäre Nachricht in die „Least Signifikant Bits (LSB)" der Daten einzubetten, die auch mit fehlerkorrigierenden Informationen versehen ist [13]. Durch die Einbettung der Daten mit einem geheimen Schlüssel lassen sich so nicht nur zufällige ungewollte Veränderungen erkennen, sondern auch gezielte Veränderungen, die durch Angreifer ausgeführt wurden [13].

Diese zwei Beispiele zeigen, dass ungewollte Veränderungen verwandt mit dem Themengebiet der gewollten Manipulationen sind, die durch einen Angreifer durchgeführt werden würden. Dadurch sind auch Wasserzeichen zur Detektion von Manipulationen für diese Arbeit relevant. Nachfolgend sollen also auch solche Algorithmen untersucht werden, welche zur Manipulationserkennung verwendet werden.

3.4.3 Wasserzeichen zur Manipulationserkennung

Im Bereich Manipulationserkennung gibt es bereits einige Forschungsarbeiten[18, 16, 11]. Dabei unterscheiden sich die Techniken abhängig von den durch sie zu erreichenden Zielen stark. Unterschiedliche Ziele führen dabei zu unterschiedlichen Anforderungen und unterschiedlichen Techniken. Nachfolgend sollen einige Ansätze vorgestellt werden.
Oben wurden bereits zwei Techniken vorgestellt, die das Wasserzeichen auf den bereits komprimierten Daten einbetten. Es gibt auch weitere Techniken, die sich die Eigenheiten des Kompressionsverfahrens zu Nutze machen. So werden zum Beispiel die „Motion-Vektoren" bzw. die „Inter-Frame-Prediction", die ein Videoencoder nach dem H.264-Standard erzeugt, benutzt, um Daten zu einzubetten[10, 11, 14]. Hierbei lassen sich ebenfalls vor allem Manipulationen während der Übertragung erkennen, da sie erst innerhalb des Encoders zum Einsatz kommen.

Neben den im Encoder arbeitenden Wasserzeichentechniken kann man auch noch die Techniken nennen, welche im unkomprimierten Bild arbeiten. Hier gibt es Techniken, die direkt auf den Pixelwerten arbeiten. Durch die gute Sichtbarkeit bzw. geringe Robustheit sind dies aber meistens eher fragile Techniken oder die Daten werden nur in bestimmten Teilen des Bildes eingebettet[15]. Robuste Techniken, die vor der verlustbehafteten Kompression eingesetzt werden, setzen häufig auf eine vorherige umkehrbare Transformation. So gibt es einige, die in der der DCT oder anderen Frequenzräumen arbeiten [16, 17, 18, 19, 12].
Eugen T. Lin et al. zum Beispiel schlagen einen Algorithmus vor, um durch die Einbettung von Wasserzeichen die Manipulation von Bildern zu erkennen, wobei hier das Wasserzeichen in die unkomprimierten Bilddaten eingebettet wird. Der Fokus der Technik liegt hierbei auf der Resistenz gegenüber verlustbehafteten Kompressionsverfahren. Dabei wird das Wasserzeichen blockweise eingebettet, wodurch sich Fehler im gesamten Bild erkennen und die Region des Fehlers sogar eingrenzen ließen[18]. Die Einbettung geschieht hierbei in den durch die DCT transformierten Bilddaten, da auch die verlustbehaftete Kompression in den auf diese Weise transformierten Daten geschieht. Dadurch treffen Eugen T. Lin et al. eine Vorhersage, welche Bereiche der Daten durch die Kompression mit höherer Wahrscheinlichkeit stark verändert werden. Die sich stark verändernden Bereiche werden bei Eugen T Lin et al. ignoriert, sodass das Wasserzeichen nur in die anderen Koeffizienten eingebettet wird. Hierbei wird der DC-Koeffizient bei der Einbettung nicht markiert, da sich eine Änderung des DC-Koeffizienten signifikant auf die Sichtbarkeit eines Wasserzeichens auswirken würde [18]. Auch höhere Frequenzen werden von dem Wasserzeichen nicht verändert, da diese mit höherer Wahrscheinlichkeit durch die verlustbehaftete Kompression stärker verändert werden [18].

3.5 ZUSAMMENFASSUNG

In diesem Abschnitt wurden Arbeiten vorgestellt, die sich ebenfalls mit der Erkennung von Fehlern in Video- oder Bildverarbeitungssystemen bzw. damit verwandten Themenfeldern beschäftigen.

Dabei fanden zuerst die Qualitätsmetriken kurze Erwähnung, die primär dazu verwendet werden, Qualität zu messen. Danach wurde eine Technik von Superiori et al. vorgestellt, die darauf basiert, eine Kombination von Differenzbildern und Syntaxanalyse einzusetzen[22]. Anschließend wurden die QR-Codes vorgestellt, welche zur Familie der Barcodes gezählt werden können[27]. QR-Codes können durch den bereits im Konzept integrierten fehlerkorrigierenden Code ohne weitere Referenzen auf Integrität geprüft werden. Indem man die künstlich erzeugten QR-Codes als zu übertragendes Video verwendet, könnten sie sich zum Testen von Videoverarbeitungssystemen eignen.

Abschließend wurde der Themenbereich der Wasserzeichen beleuchtet, wobei unterschiedliche Einbettungsmöglichkeiten vorgestellt wurden. Je nach Anwendungszweck unterscheiden sich auch die Einbettungstechniken. Anschließend wurden drei Klassen von Wasserzeichen vorgestellt: die fragilen, die robusten und die semi-fragilen. Die drei unterschiedlichen Klassen können zu unterschiedlichen Zwecken eingesetzt werden. Die robusten Wasserzeichen sollen widerstandsfähig gegen alle Arten von Veränderung sein. Fragile Wasserzeichen dagegen sollen selbst bei kleinen Veränderungen kaputtgehen. Semi-fragile Wasserzeichen kombinieren die Eigenschaften der beiden vorgenannten.
Eine Möglichkeit, um Daten mit einer gewissen Robustheit einzubetten, ist die Technik des „Spreadings“ (Streuens) [7, p. 279ff], welche durch das Verteilen der einzubettenden Informationen über das gesamte Signal eine Robustheit gegen punktuelle Veränderungen erreichen sollte.

Zudem wurde herausgestellt, dass es unterschiedliche Ansätze gibt, Fehler und Manipulationen durch Wasserzeichen zu erkennen. Es wurden zwei Fehlerarten und ihre Erkennungsmöglichkeiten vorgestellt. So gibt es Ansätze, die die Erkennung von Übertragungsfehlern zum Ziel haben und Ansätze, die zur Erkennung von Bildfehlern verwendet werden. Für die Erkennung von Übertragungsfehlern findet das Einbetten der Wasserzeichen häufig im Encoder statt [13][20]. Für die Erkennung von Veränderungen und Manipulationen auf Bildebene wird das Wasserzeichen häufig in die vorher transformierten Daten eingebettet [16, 17, 18, 19, 12].

4 EXTERNE ANFORDERUNGEN

EXTERNE ANFORDERUNGEN

Die Arbeit ist in Kooperation mit der LogMeIn Inc., in Deutschland vertreten durch die GetGo Germany GmbH, entstanden. Dabei hat die Firma eigene Anforderungen herausgearbeitet, die zusätzlich zur Aufgabenstellung als weitere Grundlage für die Arbeit dienten. Diese Anforderungen sollen im folgenden Abschnitt dargelegt werden.

4.1 KONTEXT

Die LogMeIn stellt als Produkt ein Video-Konferenzsystem und weitere Video-Kollaborationstools zur Verfügung. Um die Datenmengen, die bei Videodaten anfallen über einen Netzwerkkanal übertragen zu können, wird das Video während der Verarbeitung codiert. Die der Codierung zugrundliegende Software wird hierbei ständig verbessert und bearbeitet. Durch eine solche Veränderung im Quellcode des entsprechenden Programms, kann es vorkommen, dass sich Fehler in der Software, als visuelle Fehler in den Videodaten manifestieren. Nicht immer reichen die vorhandenen automatisierten Test- und Überprüfungsmechanismen aus, um die Fehler zweifelsfrei zu erkennen, sodass es vorkommt, dass eine Person das Video einer manuellen visuellen Prüfung unterziehen muss. Da dies zeitaufwendig sein kann, hat die Firma Bedarf an einem automatischen Ende-Zu-Ende-Testsystem, welches es erlaubt, ein durch Codierung veränderte Video auf Fehler zu überprüfen.

4.2 ANFORDERUNGEN AN DAS SYSTEM

Es soll das Konzept eines Testsystems erarbeitet werden, welches es ermöglicht, verschiedene Fehler, die im Rahmen eine Video-Konferenzsystems auftauchen, zu erkennen. Welche Fehler dies genau umfasst, wird in Abschnitt 4.2.1 definiert. Dabei soll eine Encoder-Decoder-Verarbeitungskette getestet werden. Die Encoder-Decoder-Verarbeitungskette soll dabei als Blackbox betrachtet werden. Die Blackbox erhält dabei als Eingabedaten, unkomprimierte Videodaten und gibt diese auch wieder in unkomprimierter Form aus. Dabei liegt der alleinige Fokus des Testsystems auf den visuellen Video-Daten, eine Analyse des Sounds oder Soundfehler spielen keine Rolle. In Abbildung 4.1 ist eine einfache Darstellung eines solchen Systems zu sehen. Im Rahmen des Konzeptes können die unkomprimierten Eingangsdaten dabei von der Technik verändert werden. Es soll sich jedoch weiterhin um Videos mit natürlichem Bildinhalt handeln. Das Testsystem soll dabei ohne ein Referenzvideo arbeiten, da im Falle eines verteilten Systems ein Referenzvideo beachtliche zusätzliche Bandbreite benötigen würde, was die Parameter des Tests beeinflussen könnte. Es steht jedoch ein externer Kanal für kleinere Datenmengen zur Verfügung.

Eine Qualitätsanalyse des Bildinhalts ist nicht Ziel der Arbeit. Der Fokus der Arbeit liegt ausschließlich auf der Fehlererkennung.

Des Weiteren soll das Testsystem ausschließlich für den internen Einsatz zu Testzwecken konzipiert werden, ein Einsatz im Produktivbetrieb findet nicht statt.

Bei der Kodierung mit dem H.264 ist der Quantisierungsparameter (QP) ein wichtiger Parameter, der Einfluss auf die Ausgabedaten hat, da er die Intensität der Quantisierung festlegt . Dabei sind für die LogMeIn QPs zwischen 20 und 40 besonders interessant.

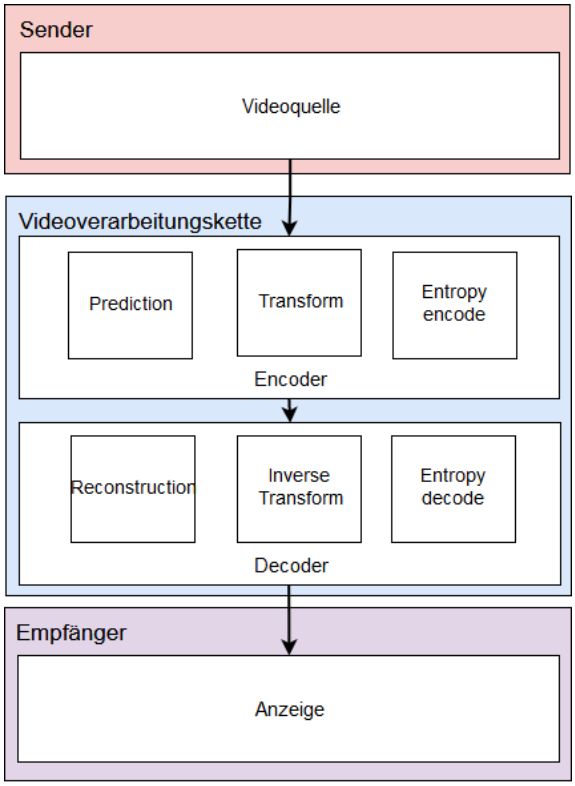


Abbildung 4.1: Schematische Darstellung, wie das zu testende System aussieht.

4.2.1 Zu erkennende Fehler

Aufgrund der Komplexität von Videoverarbeitungsketten, sind auch die in ihnen auftretenden Fehler vielzählig. Daher wurde in Kooperation mit der LogMeIn einige Fehler herausgearbeitet, die in ihren Fehlerberichten besonders häufig vorkommen.

Blockfehler: Unter Blockfehlern werden solche Fehler zusammengefasst, in denen rechtecksförmige Bereiche des Bildes andersfarbig dargestellt sind, besonders häufig kommt es vor das diese Bereiche schwarz oder grünfarbig sind. Je nachdem welche der Farben im gewählten Farbraum durch die 0 repräsentiert werden.

Croppingfehler: Als Croppingfehler werden solche Fehler bezeichnet, bei denen sich entlang des Randes eines Videos ein einfarbiger Balken bildet. In den meisten Fällen ist dieser Balken ebenfalls schwarz oder grün, dargestellt aus denselben Gründen wie schon bei den Blockfehlern dargestellt. Dies kommen zustanden, da viele Kompressionsverfahren mit festen Blockgrößen arbeiten. Je nach Auflösung lässt sich das Bild jedoch nicht restlos durch diese Blockgröße teilen, sodass das Bild um die notwendigen Pixel erweitert wird. Die Informationen über die Erweiterung wird anschließend in Metadaten gespeichert. Werden diese Metadaten nun nicht oder Fehlerhafte errechnet oder übertragen kann es sein, dass die Bilderweiterung mit angezeigt wird.

Delay: Unter Delay werden Verzögerung zwischen dem Absenden der Eingangsdaten und dem Erhalten der Ausgangsdaten bezeichnet. Dabei ist eine gewisse Verzögerung immer vorhanden, prägt sich dieser jedoch zu stark aus, soll dies vom System erkennbar sein.

Jitter: Unter Jitter wird verstanden, dass sich die Verzögerungen der Bilder sehr stark voneinander unterscheiden. Dass die einzelnen Bilder als unterschiedliche lange brauchen um

durch die Verarbeitungskette zu gelangen. Sollte dieser Unterschied sich zu signifikant ausbilden soll dies ebenfalls erkennbar sein.

Paketverlust: Unter Paketverlust wird das Fehlen von einzelnen oder mehreren Netzwerkpaketen verstanden. Das Fehlen solcher Netzwerkpakete soll ebenfalls erkennbar sein. Da die Aufgabenstellung vorzieht die Verarbeitungskette als Blackbox zu betrachten, ist das Erkennen von Paketverlust nur dann gefordert, wenn es einen visuellen Einfluss auf das Bild hat.

5 ANFORDERUNGSANALYSE

ANFORDERUNGSANALYSE

Im folgenden Kapitel werden aufbauend auf den Vorgaben des Partners die Aufgaben, die durch das Testsystem erreicht werden sollen, erarbeitet. Als Erstes wird hierbei die Problemstellung detailliert analysiert. Anschließend werden, darauf aufbauend, die zu erreichenden Ziele definiert.

5.1 ZIELSETZUNG

Ziel der Arbeit ist die Entwicklung eines prototypischen Systems (nachfolgend „System" genannt), welches im Rahmen eines Integrationstests eine abgeschlossene Video-Verarbeitungskette, bestehend aus Encoder und Decoder, automatisch ohne menschliches Zutun auf Fehler überprüfen kann. Die zu erkennenden Fehler sind dabei in Kapitel 4.2.1 bereits definiert. Aufgrund der Abgeschlossenheit des Systems kann die Fehlererkennung ausschließlich anhand der erhaltenen Ausgabedaten erfolgen.
Auch wenn es zulässig ist, die Eingabedaten zu verändern, ist eine zentrale Anforderung, dass es sich bei den Eingabedaten um weitestgehend natürliche Videosequenzen handelt. Im Folgenden wird diese Zielsetzung auf einzelne zu erreichende Ziele heruntergebrochen.

5.2 ANFORDERUNGSDEFINITION

Im Mittelpunkt der Anforderungen steht die Fähigkeit des Systems, fehlerhafte Veränderungen der Ausgabedaten zu erkennen. Diese Fehler entstehen dabei durch Fehler im System, welche sowohl Übertragungs- als auch Softwarefehler sein können.
Auch wenn das Video durch die Videoverarbeitungskette Kompressionsartefakte enthält, soll die Fehlererkennungseigenschaft dabei nicht durch die Videokompression eingeschränkt sein. Zusätzlich soll das System weiterhin auf natürlichen Videosequenzen arbeiten. Das Verwenden von künstlich erzeugten Videodaten ist nicht erlaubt. Eine Veränderung des Eingangsvideos ist im Rahmen der Aufgabenstellung explizit erlaubt. Im Rahmen dieser Arbeit wurden aus diesen Anforderungen drei übergreifende Ziele, im Folgenden als Metaziele bezeichnet, definiert. Diese lauten wie folgt:

Robustheit: Die Eigenschaften des Systems die weiteren Metaziele zu erfüllen, auch nachdem die Daten durch eine Videoverarbeitungskette verändert wurden.

Fehlererkennung: Die Eigenschaft des Systems, Fehler innerhalb der Videoverarbeitungskette anhand der Ausgabedaten zu erkennen.

Visueller Einfluss: Die visuell wahrnehmbaren Veränderungen sollen minimiert werden.

5.2.1 Metaziele

Nachfolgend werden die im vorgehenden Abschnitt definierten Metaziele noch einmal detailliert beleuchtet.

Robustheit

Bei moderner Videokompression gibt es eine Vielzahl von Operationen, wie es auszugshaft in Kapitel 2.3 dargestellt ist. Zusätzlich gibt es noch eine Vielzahl optionaler Operationen und Werkzeuge, die in dem Kapitel nicht aufgeführt sind[3, p. 92].
Diese Operationen umfassen neben verlustlosen Kompressionsverfahren auch verlustbehaftete Kompressionsverfahren, die teilweise signifikante Veränderungen am Bild zur Folge haben können. Trotz dieser verlustbehafteten Kompression und der damit verbundenen Veränderung des Bildes, soll die korrekte Funktionsweise des Systems weiterhin gegeben sein.
Videokompressionsstandards sind vielzählig und unterschiedlich, daher soll im Rahmen dieser Arbeit eine Auswahl getroffen werden, anhand derer das entwickelte Konzept später gemessen werden soll. In einer Vielzahl von Hardware-Komponenten ist Hardware-Unterstützung für einige Operationen des H.264-Standards eingebaut. So hat Intel schon in der Sandy Bridge Architektur (2011) Hardware-Unterstützung für Teile des H.264 eingebaut, andere Codecs wie beispielsweise der VP9 im Desktopbereich, haben häufig erst in späteren Modellen der Skylake Architektur (2015) Hardware-Unterstützung erhalten[24]. Nicht zuletzt deswegen gehört er in einigen Untersuchungen immer noch zum am weitesten verbreiteten Kompressionsstandard[25]. Im Rahmen dieser Arbeit wird daher als Kompressionsstandard der H.264/AVC verwendet.

Fehlererkennung

Das System soll im Rahmen eines Integrationstests Artefakte in den Ausgabedaten, die durch Fehler in der Videoverarbeitungskette entstehen, erkennen können.
Aufgrund der Komplexität moderner Videoverarbeitungssysteme, ist die Anzahl der möglichen Fehler groß, daher wurde in Kapitel 4.2.1 bereits eine Auswahl vorgegeben. Eine genaue Beschreibung der einzelnen Kategorien und eine detaillierte Aufschlüsselung der Fehler findet sich in Abschnitt 5.2.2.

Visueller Einfluss

Das Verändern der Bilddateien ist durch die Aufgabenstellung ausdrücklich erlaubt, als zentrale Anforderung wurde jedoch zusätzlich festgelegt, dass die Eingangsdaten weiterhin einem natürlichen Video entsprechen sollen. Das künstliche Erzeugen von Videoinhalten ist daher nicht erlaubt.
Damit wird als drittes Metaziel der visuelle Einfluss definiert. Die Veränderungen am Video sollen dabei soweit minimiert werden, dass das Video weiterhin als natürliches Video definiert werden kann, der Inhalt des Videos sollte also weiterhin gut erkennbar sein.

5.2.2 Definition der Fehlerklassen

Die durch das System zu erkennenden Fehler wurden bereits in Kapitel 4.2.1 definiert. Im folgenden Abschnitt werden die Fehler noch einmal detailliert behandelt und spezifiziert. Dabei wurden die Fehler zuerst in drei Kategorien eingeteilt: Bildfehler, zeitliche Fehler, Bitstromfehler. Die Kategorien und die diesen zugeordneten Fehler sind nachfolgend beschrieben.

Bildinhaltsfehler

Als Bildinhaltsfehler werden im Rahmen dieser Arbeit solche Fehler bezeichnet, bei denen einzelne oder mehrere Pixel in ihrer Farbgebung stark verändert werden. Im Rahmen dieser Arbeit wird sich dabei auf Blockfehler und Croppingfehler konzentriert. Diese beiden werden folgend noch einmal beschrieben:

Blockfehler: Dies umfasst Fehler, bei denen rechteckförmige Regionen farblich verändert dargestellt sind. Die so veränderten Blöcke erscheinen abhängig vom verwendeten Farbraum und dem spezifischen Fehler häufig als einfarbige schwarze bzw. grüne Blöcke. Die Position der Fehler kann dabei zufällig sein.
Zustande kommen diese Blockfehler durch verschiedene Fehler im Kompressionsvorgang oder durch fehlerhaft übertragene Daten, sodass für bestimmte Regionen falsche oder unvollständige Farbinformationen vorliegen. Als Folge dessen besitzen die fehlerhaften Regionen häufig als Pixelwerte den Wert null, was abhängig vom gewählten Farbraum als grüner oder schwarzer Bereich dargestellt wird. Ein Beispiel für einen solchen Fehler ist in Abbildung 5.1a dargestellt.

Croppingfehler: Dies beschreibt Fehler, bei denen einfarbige Bereiche entlang eines Bildrandes entstehen. Diese Fehler kommen zustande, wenn Metainformationen, in denen zusätzliche Informationen zur Auflösung gespeichert sind, fehlen oder fehlerhaft sind.
Viele Kompressionsverfahren arbeiten mit festen Blockgrößen. Beliebige Auflösungen lassen sich jedoch nicht immer problemlos in festen Blockgrößen aufteilen. Um dennoch Videos mit beliebigen Auflösungen durch beliebige Blockgrößen abbilden zu können, werden die Auflösungen häufig um die nötige Anzahl an Pixeln erweitert. Zusätzliche Metainformationen, die die Anzahl der so entstandenen überflüssigen Pixel speichern, werden zusätzlich zu den Bilddaten an den Decoder weitergegeben, der die Bilder wieder richtig zuschneidet. Sollten diese Metainformationen Fehler enthalten, wird das Bild nicht bzw. falsch beschnitten. Ein solcher Fehler ist in Abbildung 5.1b illustriert.

(a) Ein Bespiel für Blockfehler.

(b) Ein Bespiel eines Croppingfehlers, wobei ein 6 Pixel dicker Streifen am rechten Rand entstanden ist.

Abbildung 5.1: Die Illustration zweier Beispiele, die als Bildinhaltsfehler erkannt werden sollen.

Zeitliche Fehler

Unter zeitlichen Fehlern sind solche Fehler zusammengefasst, die die zeitliche Abfolge der Einzelbilder oder ihre Reihenfolge stören.
Für diese Arbeit sind dabei zwei Fehlerklassen interessant: zum einen Jitter-Fehler und zum anderen Delay. Beide Fehlerklassen werden im Folgenden kurz erläutert.

Abbildung 5.2: Eine Illustration, wie sich wiederholter Paketverlust auswirken kann.

Jitter: Bei Jitter werden die zeitlichen Abstände zwischen den Bildern verändert.
Bei fehlerfreien Videos haben die angezeigten Bilder meist einen definierten Abstand zueinander. Dabei kann zwischen festen und variablen Bildwiederholungsraten unterschieden werden. Bei festen Bildwiederholungsraten werden die Bilder in gleichmäßigen Abständen nacheinander angezeigt. Bei variablen Bildwiederholungsraten variiert der Abstand zwischen den einzelnen Bildern. Unabhängig von der Bildwiederholungsrate können Fehler innerhalb einer Videoverarbeitungskette dazu führen, dass sich die Abstände zwischen den Bildern verändern. Bei starken Veränderungen erscheint das Video für einen Betrachter nicht mehr als flüssig.

Delay: Delay ist der zeitliche Abstand zwischen dem Absenden der Daten auf Senderseite und dem Empfangen der Daten auf Empfängerseite. Delay ist bei Netzwerkübertragung immer gegeben. Im Fehlerfall kann es jedoch dazu kommen, dass das Delay überdurchschnittlich groß wird, sodass die Daten vom Sender zum Empfänger länger brauchen, als üblich.

Bitstromfehler

Unter Bitstromfehlern werden im Rahmen dieser Arbeit solche Fehler zusammengefasst, die durch fehlerhafte Netzwerkübertragung entstehen. Die externen Anforderungen in Kapitel 4.2.1 beschränken sich hierbei auf eine Fehlerklasse: Paketverlust.

Paketverlust: Von Paketverlust spricht man, wenn während der Übertragung einzelne oder mehrere Pakete verloren gehen. Ein Beispiel, wie sich ein solcher Fehler im Bild ausprägen kann, ist in Abbildung 5.2 illustriert.

5.3 ZUSAMMENFASSUNG

In dieser Arbeit soll im Rahmen eines Integrationstests ein prototypisches System entwickelt werden, welches es erlaubt, eine Videoverarbeitungskette automatisiert auf Fehler zu testen. In diesem Kapitel wurde ausgehend von der Aufgabenstellung und den Vorgaben des externen Partners eine Zielsetzung definiert und herausgearbeitet, welche Anforderungen an ein solches System gestellt werden. Es wurden drei zentrale Ziele - im Rahmen dieser Arbeit als Metaziele bezeichnet - herausgearbeitet:

- Robustheit

- Fehlererkennung
- Visueller Einfluss

Robustheit definiert hier die Fähigkeit des Systems, uneingeschränkt, trotz der durch die Kompressionsverfahren entstehenden Bildveränderungen, zu arbeiten. Fehlererkennung ist das Ziel, Fehler im System zu erkennen, sofern sich diese auf den Ausgabedaten visuell manifestieren. Der visuelle Einfluss bezeichnet hierbei die Eigenschaft des Systems, die Eingabedaten visuell möglichst wenig zu verändern, so dass der Bildinhalt weiterhin gut erkennbar bleibt.

Im Rahmen der Fehlererkennung wurden auch die Fehlerklassen definiert, die durch das System erkannt werden sollen:

- Bildinhaltsfehler
- Zeitliche Fehler
- Bitstromfehler

In der folgenden Tabelle sind die Ziele noch einmal aufgeführt.

ID	Name	Beschreibung
	Allgemeine Ziele	
A1	Robustheit	Die Eigenschaften des Systems, die weiteren Metaziele zu erfüllen, auch nachdem die Daten durch eine Videoverarbeitungskette verändert wurden.
	Fehlererkennung	
P1	Croppingfehler	Die Eigenschaft des Systems, Croppingfehler zu erkennen.
P2	Blockfehler	Die Eigenschaft des Systems, Blockfehler zu erkennen.
T1	Jitter	Die Eigenschaft des Systems, Jitter im Video zu erkennen.
T2	Delay	Die Eigenschaft des Systems, Delay zu erkennen.
B1	Paketverlust	Die Eigenschaft des Systems, die visuellen Ausprägungen von Paketverlust zu erkennen.
	Visueller Einfluss	
V1	Natürliches Video	Der Bildinhalt der Eingabevideos ist weiterhin gut erkennbar.

Tabelle 5.1: Die aus der Anforderungsanalyse herausgearbeiteten Ziele

6 KONZEPT

KONZEPT

Im folgenden Kapitel wird aufbauend auf der Anforderungsanalyse aus Kapitel 5 ein Konzept entwickelt, welches es erlauben soll, die Ausgabedaten einer abgeschlossenen Videoverarbeitungskette automatisch auf die in der Anforderungsanalyse definierten Fehler zu testen. Das Kapitel ist dabei wie folgt aufgebaut:
Zuerst wird ausgehend von den verwandten Arbeiten eine grundlegende Idee skizziert, darauf aufbauend wird, der Algorithmus entwickelt. Anschließend werden die Details der Umsetzung beschrieben. Abschließend wird detailliert analysiert wie mit dem so entstehenden Algorithmus die Fehlererkennung funktioniert.

6.1 BASIS

In Kapitel 3 wurden bereits Ansätze zum Testen von Video- und Bildverarbeitungssystemen vorgestellt. Nachfolgend werden die unterschiedlichen Ansätze aufgenommen und überprüft, ob sie anwendbar auf die Anforderung aus Kapitel 5 sind.

6.1.1 Qualitätsmetriken

Video- und Bildqualitätsmetriken können die Qualität eines gegebenen Videos ermitteln. Dabei gibt es sowohl referenzbasierte Ansätze, Ansätze, die ohne Referenz auskommen und Ansätze die mit Teilinformationen der Referenz arbeiten. Auch wenn die Qualitätsmessung ein verwandtes Themenfeld der Fehlererkennung ist, werden Qualitätsmetriken immer auch den Qualitätsverlust, der durch die Kompression entstanden ist, abbilden. Da Qualitätsanalyse laut Aufgabenstellung nicht Teil der Aufgabe ist, eignen sich Qualitätsmetriken für die gegebene Aufgabenstellung nicht.

6.1.2 Integritätsprüfung

Fehlererkennung könnte auch durch Integritätsprüfung eines bekannten oder rekonstruierbaren Bildinhaltes erreicht werden. Der in Kapitel 3 vorgestellte QR-Code könnte durch den eingebetteten fehlerkorrigierenden Code eine Beschädigung seiner selbst erkennen, lokalisieren und zu einem gewissen Grad rekonstruieren[2, p. 198]. Allerdings würde der Einsatz des QR-Codes bedeuten, dass das Video durch künstlich erzeugte Bildinhalte ersetzt werden müsste, was durch die Aufgabenstellung ausgeschlossen wird.

6.1.3 Wasserzeichen

Um möglichst unsichtbar zusätzliche Informationen in Multimediadaten unterzubringen, wurden im Kapitel 3 Wasserzeichen vorgestellt. Wasserzeichen existieren sowohl in robuster, fragiler und semi-fragiler Form[19]. Fragile Wasserzeichen werden dazu eingesetzt, die Integrität eines gegebenen Bildes zu verifizieren, indem das Wasserzeichen kaputtgeht, sobald kleinste Veränderungen an den Bilddaten vorgenommen wurden. Im Gegensatz dazu werden robuste Wasserzeichen entwickelt, um auch gegenüber starken Veränderungen am Bild stabil zu sein. Semi-fragile Wasserzeichen kombinieren die Eigenschaften beider genannter Techniken, sie sind

robust gegenüber einigen Veränderungen und fragil gegenüber anderen. Ein semi-fragilen Wasserzeichen könnte sowohl verlustbehafteten Kompression überstehen und gleichzeitig durch Fehler zerstört werden. Mit einem Detektionsmechanismus der prüft ob das Wasserzeichen weiterhin intakt ist könnten dies ein vielversprechender Ansatz sein, um die in Kapitel 5 herausgearbeiteten Anforderung zu erfüllen.

6.1.4 Entwurf

In Kapitel 5.2.2 wurden 3 Metaziele definiert, die wie folgt lauten: Robustheit, Fehlererkennung und visueller Einfluss. Robustheit bedeutet, dass das System auch funktionsfähig sein soll, nachdem die Videodaten verlustbehaftet komprimiert wurden. Fehlererkennung definiert das Ziel Fehler, die bei der Verarbeitung entstehen zu erkennen. Visueller Einfluss definiert, dass durch das hier entwickelte Testsystem keine zu starken Veränderungen am Bild vorgenommen werden dürfen. Aufbauend auf diesen Metazielen und den verwanden Arbeiten, soll nun ein System Entworfen werden.

Um die in der Anforderung definierten Metaziele zu erfüllen, werden in dieser Arbeit die Integritätsprüfung der QR-Codes und die semi-fragilen Wasserzeichens kombiniert. Dabei wird auf den folgenden Seiten ein semi-fragiles Wasserzeichen entwickelt, welches robust gegenüber der Videoverarbeitungskette und fragil gegenüber Bildinhaltsfehlern ist. Durch den Einsatz von Wasserzeichen können die Informationen in ein bestehendes Video eingebettet werden und so bleibt, im Gegensatz zu QR-Codes, ein natürliches Bild erhalten. Mit diesem Ansatz können also alle Metaziele erreicht werden.

In Kapitel 3 wurden bereits einige Techniken zur Wasserzeicheneinbettung benannt und erklärt, hiervon bieten sich im Rahmen dieser Arbeit insbesondere jene Verfahren an, welche auf Einzelbildern arbeiten: Sie ermöglichen es, unabhängige Wasserzeichen in jedes Bild des Videos einzufügen und Fehler somit bildgenau zu detektieren.
Im Rahmen dieser Arbeit wird angenommen, dass die in Kapitel 5.2.2 definierten Fehler, insbesondere Blockfehler, an zufälligen Positionen auftreten. Daraus folgt, dass nur solche Ansätze anwendbar, die das Wasserzeichen in allen Teilen des Bildes einbetten.

Eugen T. Lin et. al. haben ein Wasserzeichen vorgestellt, welches sie in die durch die DCT transformierten Bilddaten einbetten[18]. Die verlustbehaftete Kompression geschieht im H.264/AVC-Standard zu einem erheblichen Teil während der Transformation und Quantisierung [3, p. 179]. Durch das Einbetten in den ebenfalls transformierten Daten lässt sich zu einem gewissen Grad vorhersagen, welche Veränderungen durch die verlustbehaftete Kompression entstehen[18]. Durch diese Vorhersage lassen sich die Daten an die erwarteten Veränderungen anpassen. So kann eine höhere Robustheit erreicht werden. Daher wird auch in der hier vorgeschlagenen Technik das Einbetten in den durch die DCT transformierten Bilddaten vorgenommen.
Außerdem bildet sich eine Veränderung eines einzelnen DCT-Koeffizienten auf mehrere Pixelwerte gleichzeitig aus. Die Annahme ist, dass durch die so entstehende Streuung die Veränderungen, die durch das Wasserzeichen entstehen, visuell weniger sichtbar sind. Die Fehlererkennung wird durch die eingebetteten Daten erreicht. So wird der fehlerkorrigierende Code eingesetzt, um zu überprüfen, ob Fehler in der Verarbeitung das Bild verändert und damit das Wasserzeichen zerstört haben. Zusätzlich wird ein Zeitstempel als Wasserzeichen eingebettet, das beim Extrahieren, mit dem Zeitstempel des Extraktionszeitpunktes abgeglichen werden kann. Indem die Differenz zwischen Einbettung und Extraktionszeitpunkt errechnet wird, lässt sich Delay erkennen. Jitter kann erkannt werden, indem die zeitlichen Differenzen der einzelnen Bilder untersucht werden. Liegt eine hohe Varianz der Delays vor bedeutet dies, dass ein Jitter im Video vorhanden ist. Bitstromfehler werden entweder dazu führen, dass einzelne Bilder fehlen oder dass das Bild stark verändert ist. Daher wird ein eindeutiges

Identifikationsmerkmal als Wasserzeichen eingebettet, um das Fehlen einzelner Bilder zu erkennen. Falls das Bild verändert sein sollte, lässt sich dies analog zu den Bildfehlern erkennen.

Um zeitliche Fehler erkennen zu können, wird zusätzlich noch ein Zeitstempel ins Bild eingebettet. So wird der Einbettungszeitpunkt und damit auch der Zeitpunkt, kurz bevor die Daten an die Videoverarbeitungskette übergeben werden, im Wasserzeichen festgehalten. Durch das Errechnen der Differenz zwischen extrahiertem Einbettungszeitpunkt und Extraktionszeitpunkt lassen sich zeitliche Fehler erkennen.

6.2 ALGORITHMUS

Im Folgenden wird ein semi-fragiles Wasserzeichen entwickelt werden, welches es erlaubt, eine Videoverarbeitungskette, bestehend aus Encoder und Decoder, auf Fehler zu testen. Dabei wird das Konzept aufbauend auf die in Kapitel 5.2 ermittelten Metaziele - Robustheit, Fehlererkennung und visueller Einfluss - entwickelt.

Das semi-fragile Wasserzeichen wird dabei robust gegenüber verlustbehafteter Kompressionen sein. Gleichzeitig wird es in der Lage sein, die Fehler, die sich in den Ausgabedaten manifestieren, zu erkennen und dabei einen geringen visuellen Einfluss auf die Daten haben.

Das hier vorgeschlagene Wasserzeichen ist an die Technik des „Streuens"(Spreading), wie von Peter Wayner erklärt [7, p. 279ff], angelehnt. Beim Streuen wird das Wasserzeichen nicht an einzelnen ausgewählten Stellen verändert, sondern die Veränderung wird über das gesamte Signal verteilt.

Bei Peter Wayners Vorschlag wird das Wasserzeichen auf ein eindimensionales Signal angewandt. Durch Absenken und Anheben der Signalkurve lassen sich so zwei unterschiedliche Werte zum Beispiel 1 und 0 kodieren. Das Anheben bzw. Absenken geschieht dabei in Relation zu einem Schwellwert T, wobei oberhalb dieses Wertes das Wasserzeichen als 1 interpretiert wird und unterhalb als 0.

Im Rahmen dieser Arbeit wird dieser Ansatz auf Bilddaten erweitert, indem die Technik des „Spreadings" auf ein zweidimensionales Signal übertragen wird. Als Trägermedium werden die durch die DCT transformierten Bilddaten in Form von DCT-Koeffizienten verwendet. Dabei werden die DCT-Koeffizienten in der Summe angehoben bzw. abgesenkt. Es wird also die Summe der DCT-Koeffizienten gebildet und durch eine auf alle Koeffizienten gestreute Veränderung wird die Summe der Koeffizienten so verschoben, dass sie über oder unter dem Schwellwert T liegt. Bei der Extraktion des Wasserzeichens kann daraufhin durch Summenbildung und Abgleichung mit dem Schwellwert T erkannt werden, ob die Summe der DCT-Koeffizienten oberhalb oder unterhalb von T liegt also ob eine 1 oder 0 kodiert wurden. Durch die verteilte Veränderung der Koeffizienten, wird das Signal stabil gegenüber verlustbehafteter Kompression. Die Annahme ist hierbei, dass der Encoder nur Teile des Blockes stark verändert, sodass sich die Veränderung auf die Gesamtsumme aller Koeffizienten nicht zu stark auswirkt. Da eine Veränderung der Summe durch Kompression aber nicht zu verhindern ist, wird die Verschiebung, die vorgenommen wird, immer auf den Wert $T \pm B$ geschoben, wobei T der Schwellwert und B ein Puffer für eventuelle Veränderung ist. In Abbildung 6.1 ist dieser Vorgang schematisch dargestellt.

Beim Auftreten von Fehlern werden sich die DCT-Koeffizienten anders und weniger gerichtet verändern als bei der verlustbehafteten Kompression. Die Hypothese ist, dass durch verlustbehaftete Kompression hervorgerufene Veränderungen durch den Puffer B abgefangen werden, während die Bildfehler die Summe so stark verändern, dass die Summe auf die jeweils andere Seite des Schwellwertes verschoben wird. Durch den Einsatz eines fehlerkorrigierenden

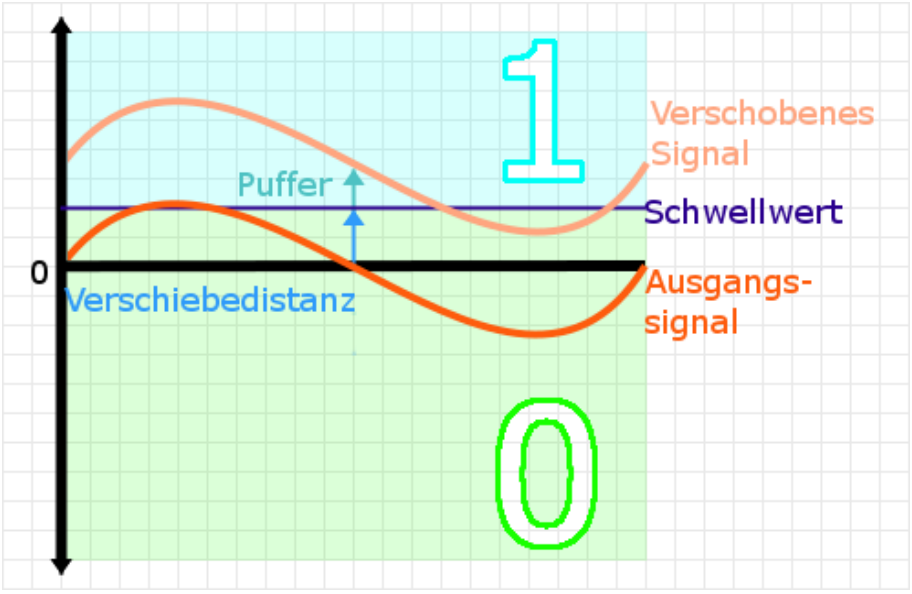


Abbildung 6.1: Schematische Darstellung, wie ein Wasserzeichen durch das Verschieben eines Signals eingebettet wird. Dabei stehen Werte unterhalb des Schwellwertes für den Wert 0 und oberhalb für den Wert 1.

Codes lässt sich anschließend erkennen, dass sich ein Bit verändert hat. So lassen sich Bildfehler durch einen Integritätscheck erkennen, während verlustbehaftete Kompression die Integrität des Wasserzeichens nicht stört. Eine Weiterentwicklung dieser Grundidee findet im Abschnitt 6.2.2 statt.

Im Folgenden wird der Aufbau des Algorithmus detailliert beschrieben.

6.2.1 Grundlagen

Im Rahmen dieser Arbeit soll ein semi-fragiles Wasserzeichen entwickelt werden, welches auf der Idee des „Spreadings“ basiert. Durch Anheben bzw. Absenken der Summe der DCT-Koeffizienten, kann ein Bit eingebettet werden.

Blockweise Einbettung

Das Wasserzeichen wird blockweise eingebettet. Dazu wird das Bild in Blöcke unterteilt und jeder Block kodiert ein Bit des Wasserzeichens. Hierzu wird jeder Block DCT-transformiert und die Koeffizienten werden so verändert, dass ihre Gesamtsumme einem gewünschten Wert entspricht. Sollte ein Bildfehler auftreten, wird sich die Summe des betroffenen Blocks verändern. Diese Veränderung wird auch den eingebetteten binären Wert verändern. Diese Veränderung ist mithilfe des fehlerkorrigierenden Codes erkennbar[2, p. 198]. So lässt sich blockgenau die Stelle erkennen, an der der Fehler aufgetreten ist. Der blockweise Aufbau ist in 6.2 noch einmal schematisch illustriert.

Bei einem blockweisen Aufbau ist ein wichtiger Parameter, wie hoch und breit ein Block ist. Nachfolgend werden die Breite und Höhe des Blockes als Blockgröße bezeichnet. Um sicherzustellen, dass bei der Einbettung und Extraktion dieselbe Blockgröße verwenden wird, wird für alle Videos eine feste Blockgröße festgelegt. Die Blockgröße verändert sich durch die

DCT-Transformation nicht, daher existieren immer so viele DCT-Koeffizienten, wie das ursprüngliche Block Pixel besitzt.
Die Robustheit der hier vorgeschlagenen Technik basiert dabei auf der Streuung der Informationen über mehrere DCT-Koeffizienten. Daher wird bei der Wahl der Blockgröße versucht, die Breite und Höhe möglichst hoch zu wählen. Gleichzeitig muss jedoch bedacht werden, dass bei einer größeren Blockgröße und gleicher Auflösung die Einbettungskapazität sinkt. Die genaue Festlegung der Blockgröße findet in Kapitel 8.1.1 statt.

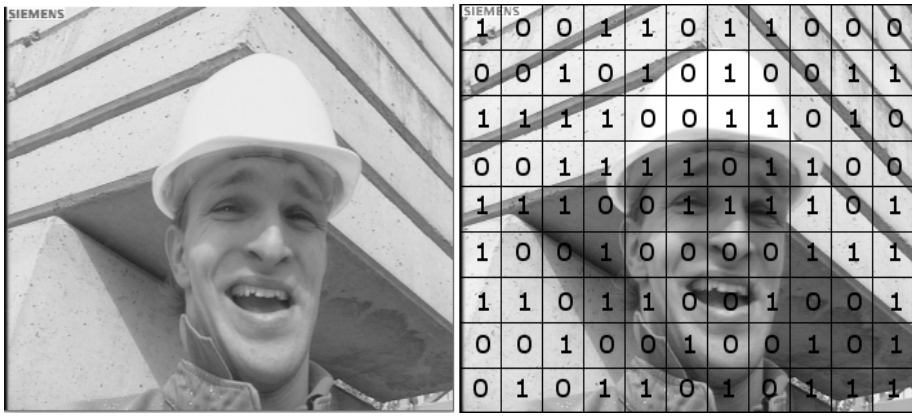


Abbildung 6.2: Schematische Darstellung, des blockweisen Aufbaus des Wasserzeichens, links das Originalbild und rechts eine Illustration, wie die Bits pro Block eingebettet werden können.

Gewichtung der Koeffizienten

Bildveränderungen treten sowohl durch verlustbehaftete Kompression, als auch durch Fehler in der Verarbeitungskette auf. Die verlustbehafte Kompression versucht dabei, visuell wichtige Informationen im Bild zu erhalten. Bildfehler hingegen verändern die visuelle Erscheinung und die Strukturen im Bild zufällig. Anhand der Art und Stärke der Veränderung lassen sich so Bildfehler von Veränderungen durch die verlustbehaftete Kompression unterscheiden.

Eugen T Lin. et al haben ein Wasserzeichen entwickelt, welches ebenfalls stabil gegenüber verlustbehafteter Kompression ist und die Erkennung von ungewollten Veränderungen im Bild erlaubt[18]. Dabei wird vorgeschlagen, die Informationen in den DCT-Koeffizienten zu verstecken, die vom Encoder mit geringerer Wahrscheinlichkeit verändert werden. Es wird also eine Art Vorhersage getroffen, welche Veränderungen durch verlustbehaftete Kompression am Bild vorgenommen werden. Weichen die Veränderungen von dem Erwarteten ab, wird angenommen, dass es sich um einen Fehler handelt. Ein ähnlicher Ansatz wird auch in dieser Arbeit verfolgt.

Eugen T. Lin et al. haben ihr Wasserzeichen dabei für den Einsatz auf Einzelbildern und dem JPEG-Codec entwickelt. In der vorgeschlagenen Technik wurde die Einbettung des Wasserzeichens auf der gleichen Blockgröße die auch der JPEG-Codecs verwendet, vollzogen. Die Entscheidung, welche DCT-Koeffizienten vom Wasserzeichen verändert werden, wurde binär getroffen. Das bedeutet das ein Koeffizient entweder verändert wurde oder nicht.
Im Unterschied zu Einzelbild-Endcodern, wie z.B. JPEG, nutzt der H.264-Standard eine Bewegungsvorhersage und kodiert lediglich den Fehler, u.a. per blockweiser DCT und anschließender Quanitisierung. Hierzu definiert der Standard eine größere Anzahl Coding-Tools und spezifiziert lediglich den zu decodierenden Bitstrom. Die H.264 Encoder-Implementierung hat somit große Freiheiten bei der Kombination der Coding Tools, was es im Vergleich zu z.B. einem JPEG Encoder ungleich schwieriger macht, Vorhersagen zur Veränderung der Daten zu treffen. Im Rahmen dieser Arbeit werden daher nicht einzelne fest definierte Koeffizienten

verändert, sondern es wird eine Gewichtungsfunktion vorgeschlagen, wobei solche Werte durch das Wasserzeichen stärker verändert werden, die voraussichtlich durch die verlustbehaftete Kompression relativ wenig verändert werden und die Werte, die mit einer höheren Wahrscheinlichkeit verändert werden, werden vom Wasserzeichen weniger stark verändert. Dadurch lässt sich die Robustheit des Wasserzeichens erhöhen. Zusätzlich soll durch die Gewichtung sichergestellt werden, dass die Veränderung der DCT-Koeffizienten proportional zu ihrem generellen Einfluss geschieht. Dadurch sollen die Veränderungen weniger Sichtbar sein. Die genaue Gewichtung wird im Rahmen der Analyse in Kapitel 8, abhängig von den Veränderungen und der durchschnittlichen Höhe der Koeffizienten, entwickelt.

Bei der hier vorgestellten Technik wird die Robustheit aus dem Streuen von Informationen über mehrere Informationsträger erreicht. Bei der Einbettung wird dabei jeder Koeffizient abhängig von einer Gewichtung verändert. Dabei wird die Matrix der DCT-Koeffizienten mit der Gewichtung multipliziert und anschließend abhängig von der Intensität der Veränderung mit der Matrix der DCT-Koeffizienten addiert.

Zusätzliches Rauschen

Wenn viele DCT-Koeffizienten null sind, können durch die Multiplikation lediglich die wenigen Koeffizienten beeinfluss werden die ungleich null sind. Daher würde in einem solchen Fall, die Veränderungen nicht mehr gestreut werden, sondern die Veränderungen wären wieder punktuell. Eine solche punktuelle Veränderung, die sich nur auf einen oder wenige Koeffizienten bezieht, kann visuell stark sichtbar sein.
Transformiert man beispielsweise einen einfarbigen Block mit der DCT entsteht eine DCT-Koeffizienten-Matrix, in der lediglich ein Wert, der DC-Wert, ungleich null ist. Dies ist in Abbildung 6.3 illustriert. Eine detaillierte Beschreibung des Einbettungsvorgangs findet sich in Abschnitt 6.2.2.

Daher wird im Fall, dass nur einer oder wenige Koeffizienten ungleich null sind eine Art künstliches Rauschen auf das Bild gerechnet. Dies ist in Abbildung 6.4 dargestellt,. Durch die so hinzugewonnen Werte für die Koeffizienten kann die Information wieder über mehrere Werte gestreut werden. Als Rauschen werden dabei die absoluten Werte einer normalverteilten normalverteilte Zufallswerte im Wertebereich -1 bis 1.[1]

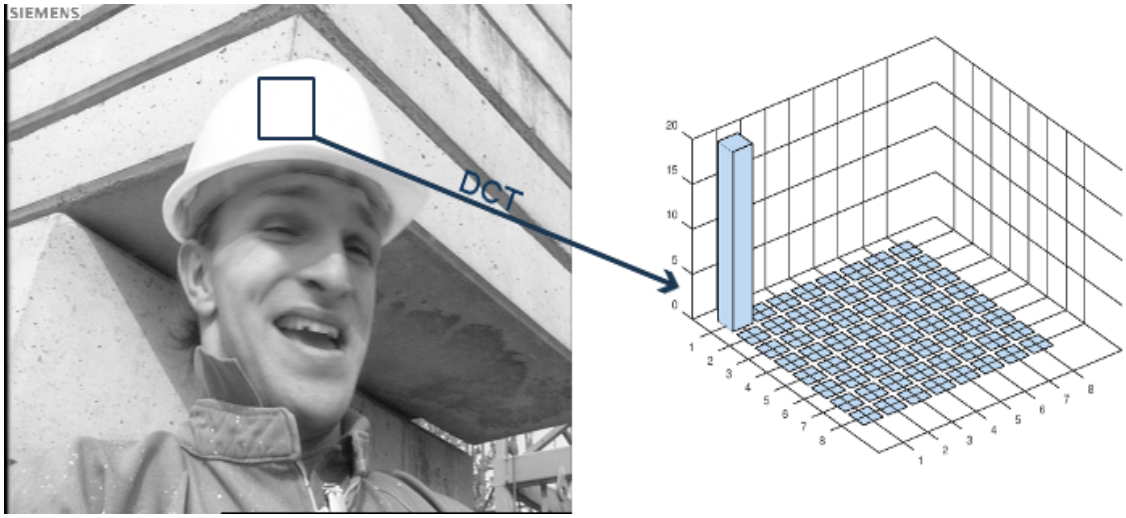


Abbildung 6.3: Illustration, wie die DCT-Koeffizienten eines einfarbigen Blocks aussehen können.

[1] Konkret wurde die Zufallswerte wie folgt generiert: abs(np.random.normal(size=(self.cb, self.cb)))

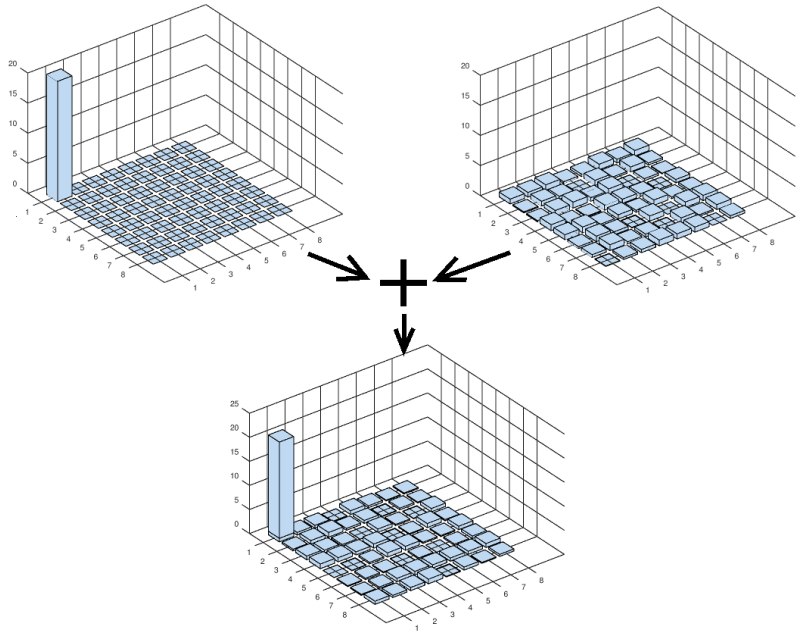


Abbildung 6.4: Illustration, wie zu strukturschwachen Bildregionen durch den Einsatz zufälliger Werte Strukturen hinzugefügt werden, um eine höhere Robustheit zu erreichen. Links oben ein mit der DCT-transformiertes Bild, rechts eine Matrix zufälliger Werte, beide addiert, ergibt dies die untere Matrix

6.2.2 Umsetzung

Auf den vorhergehenden Seiten wurde ein Wasserzeichen-Algorithmus entwickelt, welcher durch Anheben oder Absenken eines Signals einen binären Wert kodiert. Das Wasserzeichen wird hierfür blockweise in die durch DCT transformierten Bilddaten eingebettet. Das Absenken bzw. Anheben des Signals geschieht, durch eine gestreute Veränderung der DCT-Koeffizienten eines Blocks. Um eine erhöhte Robustheit zu erreichen wird zusätzlich eine Gewichtung eingeführt, wobei sich bei der Kompression stark verändernde Koeffizienten weniger gewichtet werden. Nachfolgend soll die konkrete Umsetzung dieser Idee als Algorithmus beschrieben werden und eventuelle Erweiterungen mit eingebracht werden.

Der Schwellwert

In Abschnitt 6.2 wurde die Grundidee, ein Wasserzeichen in ein Signal einzubetten, indem das Signal abgesenkt bzw. angehoben wird, vorgestellt. Das Wasserzeichen bettet so binäre Werte ein. Welcher binäre Wert eingebettet wird, wird dabei anhand eines Schwellwertes bestimmt, Werte oberhalb dieses Schwellwertes entsprechen dabei einer 1 bzw. 0, Werte unterhalb entsprechen dem jeweils anderen Wert. Im Folgenden wird die Idee eines solchen Schwellwertes, mit dem Ziel den visuellen Einfluss weiterhin zu reduzieren, weiterentwickelt.

Die DCT-Koeffizienten sind abhängig vom Bildinhalt, daher können die Summen der DCT-Koeffizienten sich bei unterschiedlichem Bildinhalt auch stark voneinander unterscheiden. Damit verändert sich auch die Distanz, um die eine DCT-Koeffizienten-Matrix verschoben werden muss. Bei der Wahl eines einzigen festen Schwellwertes ergibt sich somit das Problem, dass die DCT-Koeffizienten je nach Bildinhalt unterschiedlich stark verändert werden müssen. Dies wird sich mit zunehmender Stärke der Änderung auch sichtbar im Bildinhalt niederschlagen. Der

visuelle Einfluss soll jedoch, wie in der Anforderungsanalyse in Kapitel 5 definiert ist, minimiert werden.
Um die Verschiebung aller möglichen DCT-Koeffizienten Matrizen zu reduzieren, werden in dieser Arbeit mehrere Schwellwerte verwendet. Dabei werden die Schwellwerte in einem gleichmäßigen Abstand zueinander definiert. Der Bereich zwischen zwei nebeneinanderliegenden Schwellwerten (diese Bereiche werden nachfolgend Buckets genannt) entspricht immer einem binären Wert. Dabei sind die Buckets in einer alternierenden Reihenfolge angeordnet, sodass zwei aneinandergrenzende Buckets nicht demselben Wert entsprechen. Bei der Verschiebung wird nun immer ein naheliegender Bucket ermittelt, der dem einzubettendem Wert entspricht. So kann die Distanz, um den verschoben werden muss, reduziert werden. Die Anordnung ist in Abbildung 6.5 illustriert. Dabei sind die Schwellwerte als schwarze gestrichelte Linien gekennzeichnet und befinden sich an den Positionen $X \cdot T$ mit $X,T \in \mathbb{N}$.

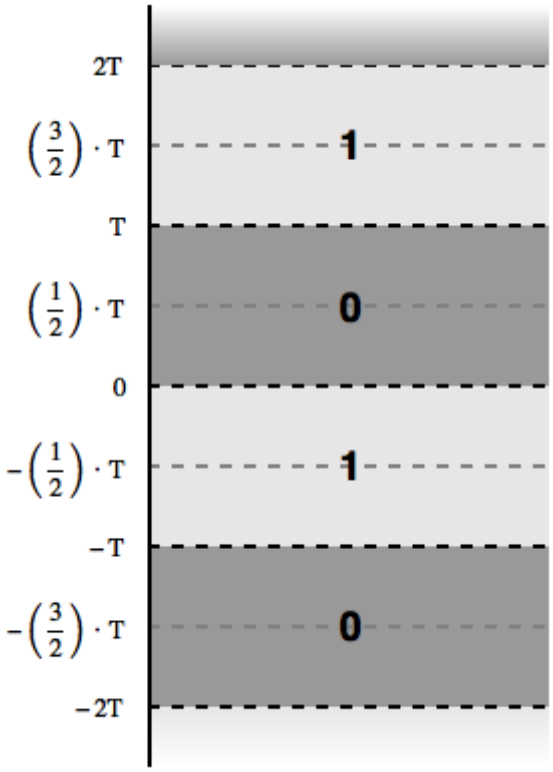


Abbildung 6.5: Die Umsetzung des Schwellwertes durch mehrere Teilabschnitte. Jeder Teilabschnitt steht für einen binären Wert.

Schon bei der ursprünglichen Vorstellung des Schwellwerts am Anfang von Abschnitt 6.2, wurde die Idee eines Puffers eingeführt. Der Puffer bietet einen gewissen Spielraum für Veränderungen, die unweigerlich durch die verlustlose Kompression entstehen. Puffer finden auch in dem Ansatz der Buckets Anwendung. Bei der Verschiebung eines Blockes, wird die Summe nicht nur an eine beliebige Stelle innerhalb eines Buckets verschoben, sondern es wird vorher, für jeden Block einzeln, ein Zielwert (im folgendem auch target genannt) festgelegt. Der target-Wert ist dabei immer der Mittelpunkt zwischen zwei aneinandergrenzenden Schwellwerten. Der Mittelpunkt ist in Abbildung 6.5 als graue gestrichelte Linie in jedem Block dargestellt. Somit entsteht ein Puffer zur oberen und unteren Grenze von $\frac{1}{2}T$

Durch die Wahl eines Schwellwertes T lässt sich die Größe der Buckets und damit auch die Verschiebungsdistanz variieren. Um einen Wert $V \mid V \in \{0,1\}$ in einen Block einzubetten, muss die Summe der DCT-Koeffizienten auf den Target-Wert verschoben werden. Das Target für einen

Block, der durch das Wasserzeichen verschoben wird, lässt sich in Abhängigkeit vom Wert T, dem einzubettenden Wert V und der summierten DCT-Koeffizienten S_c wie folgt berechnen

$$target = \begin{cases} \left(\left(\left\lfloor \frac{S_c}{2*T} \right\rfloor * 2 * T\right) + T\right) - \frac{T}{2}, & \text{wenn V=0} \\ \left\lfloor \frac{S_c + T}{2*T} \right\rfloor * 2 * T - \frac{T}{2}, & \text{anderenfalls} \end{cases} \tag{6.1}$$

Die Extraktion eines so eingebetteten Wertes V kann mit folgender Formel berechnet werden:

$$V \equiv \left\lfloor \frac{S_c}{T} \right\rfloor \pmod 2 \tag{6.2}$$

Die Verschiebung

Die konkrete Einbettung des Wasserzeichens wird durch das Verschieben der durch die DCT transformierten Bilddaten erreicht. Die konkrete Umsetzung der Verschiebung kann mit folgendem Algorithmus beschrieben werden:

1. DCT-Transformation des Blocks
2. Ermittlung des Target-Wertes
3. Summierung der Koeffizienten
4. Berechnung der Distanz zwischen Target und der Summe der Koeffizienten
5. „Verschieben“ der Koeffizienten.
6. Inverse Transformation des Blocks

Dabei werden die Bilddaten zu allererst durch die DCT transformiert. Anschließend wird der Zielwert ermittelt, auf den die summierten Koeffizienten verschoben werden sollen. Daraufhin wird die Distanz zwischen der momentanen Summe und dem Target ermittelt. Anschließend wird der Block gewichtet verschoben, indem die errechnete Veränderung abhängig von einer Gewichtungsmatrix über die Koeffizienten gestreut wird. Abschließend wird der Block durch die die inverse DCT-Transformation zurück transformiert.

Um die gewichtete Verschiebung zu realisieren, wird das transformierte Bild als Matrix betrachtet, wobei jeder DCT-Koeffizient einem Wert in der Matrix entspricht. Die Matrix der DCT-Koeffizienten wird nachfolgend mit **C** bezeichnet. Die Gewichtung wird ebenfalls in Form einer Matrix beschrieben, wobei die Gewichtungsmatrix dieselben Dimensionen hat wie **C**. Jeder Eintrag in der Gewichtungsmatrix steht für einen Gewichtungsfaktor eines DCT-Koeffizienten. Die Gewichtungsmatrix wird folgend mit **W** bezeichnet.

Angelehnt an die Verschiebung von Peter Wayner[7, p.286] erfolgt das Verschieben der Blöcke wie folgt:

$$\begin{aligned} c_{sum} &= \sum | \boldsymbol{C} | \\ d &= \textbf{target} - c_{sum} \\ C_{new} &= \boldsymbol{C} + d \cdot \frac{|C| \cdot W}{c_{sum}} \end{aligned} \tag{6.3}$$

Der so verschobene Block C_{new} wird nun durch die inverse DCT-Transformation zurücktransformiert. Die so erhaltenen Bilddaten werden anschließend durch die Videoverarbeitungskette verarbeitet und mit der bereits in Abschnitt 6.2.2 beschriebenen Formel 6.2 extrahiert. Die genaue Gewichtung wird im Rahmen der Analyse untersucht und festgelegt. Dabei wird untersucht, welche Koeffizienten durchschnittlich höher sind und welche Koeffizienten weniger stark verändert werden. Der Wert Target entspricht dabei dem bereits im Abschnitt 6.2.2 erwähnten Zielwert, auf den der Block verschoben werden soll.
Andere Arbeiten haben gezeigt, dass bereits relativ schwache Veränderungen des DC-Wertes sich stark auf die visuelle Erscheinung des Bildes auswirken können[18], daher wird keine Veränderung des DC-Wertes vorgenommen und die Verschiebung damit auf alle Koeffizienten außer den DC-Wert gestreut.

6.3 FEHLERERKENNUNG

Auf den vorhergehenden Seiten wurde erklärt, wie mithilfe des Wasserzeichens Informationen ins Bild eingebettet werden können. Im folgenden Abschnitt wird nun erklärt, welche Informationen durch das Wasserzeichen eingebettet werden und wie durch diese Informationen die Fehlererkennung realisiert wird.

Die Informationen, die eingebettet werden, sind dreigeteilt. Die drei Teile lauten wie folgt:

- Zeitliche Informationen
- Bildidentifikations-Informationen
- Fehlerkorrigierender Code

Die zeitlichen Informationen werden dazu verwendet die in Kapitel 5.2.2 definierten zeitlichen Fehler zu erkennen. Die Bildidentifikations-Informationen geben jedem Bild ein eindeutiges Erkennungsmerkmal, welches das Erkennen von fehlenden Bildern ermöglicht. Fehlende Bilder können abhängig von der verwendeten Codierung unter Umständen durch Paketverlust hervorgerufen werden. Der fehlerkorrigierende Code wird eingesetzt, um Fehler auf Bildebene zu erkennen. Fehler auf Bildebene werden aufgrund der semi-fragilen Eigenschaft des Wasserzeichens die Integrität der eingebetteten Daten stören, wie im Abschnnitt 6.2.1 erklärt. Der fehlerkorrigierende Code erlaubt es, die Integrität zu prüfen und so fehlerhafte Blöcke zu erkennen und zu lokalisieren.

6.3.1 Zeitliche Informationen

Bei Jitter und Delay variiert die Zeit zwischen dem Eingang der Daten in die Videoverarbeitungskette und der Ausgabe derselben Daten aus der Videoverarbeitungskette. Daher muss, um diese Fehler durch das Prüfsystem erkennen zu können bekannt sein, zu welchem Zeitpunkt die Daten in die Videoverarbeitungskette eingegangen sind und zu welchem Zeitpunkt sie wieder herausgekommen sind.
Damit dem Prüfsystem diese Informationen vorliegen, wird beim Einbetten der zu dem Zeitpunkt aktuelle Zeitstempel eingebettet. Durch den Vergleich des Zeitstempels aus den Bilddaten und dem zum Zeitpunkt der Extraktion aktuellem Zeitstempel kann so eine bildgenaue Aussage getroffen werden, wie lange die Verarbeitung an dieser Stelle gedauert hat. Der Zeitstempel wird als binäre Zeichenfolge in das Bild eingebettet, wobei ein Bit pro Block eingebettet wird.

Delay und Jitter können wie in Kapitel 4.2.1 getrennt voneinander betrachtet werden. Sie lassen sich wie folgt detektieren: In beiden Fällen wird die Differenz zwischen dem eingebetteten Zeitstempel und dem Zeitstempel des Extraktionszeitpunktes errechnet. Somit erhält man die Verzögerung für jedes Bild. Dies kann nun bildgenau als Delay verwendet werden oder es wird der Durchschnitt berechnet um einen einheitlichen Wert für den Delay zu erhalten.
Jitter kann erkannt werden, indem die Verzögerungen der einzelnen Bilder untereinander verglichen werden. Variieren diese stark bedeutet dies, dass Jitter vorhanden ist.

Theoretisch ist die Zeit die das Video innerhalb der Verarbeitungskette verbringt nach oben hin nicht begrenzt. Durch andere Prüfmechanismen wie Netzwerk-Zeitüberschreitung oder das Untersuchen der allgemeinen Verarbeitungszeit lassen sich besonders starke zeitliche Verzögerungen, auch ohne die hier vorgestellte Technik, erkennen. Der Zeitraum der durch das Prüfsystem überprüfbar ist kann daher auf ein nötiges Maß reduziert werden. Durch eine Verringerung des abzudeckenden Zeitraums verringern sich auch die einzubettenden Informationen. Die so freigewordenen Kapazitäten können für eine höhere Redundanz des fehlerkorrigierenden Codes verwendet werden und damit kann eine höhere Robustheit erreicht werden. Als Zeitstempel wird ein Zeitstempel verwendet, der die vergangenen Sekunden seit der Epoche hochzählt [36]. Um die Länge des Zeitstempels zu verringern werden die Letzten Bits des Zeitstempels extrahieren. Somit lässt sich ein spezifischer Zeitraum abdecken. Nachfolgend soll nun ein sinnvoll abzudeckender Zeitraum gefunden werden.

Bereits relativ geringe zeitliche Verzögerungen bei Live-Konferenzen können auf Nutzer störend wirken[9], daher muss die Auflösung des Zeitstempels auch bereits geringe zeitliche Veränderungen abbilden können. Im Rahmen dieser Arbeit wurde als kleinste darstellbare Einheit eine Millisekunde gewählt.

Als maximal erkennbarer Zeitraum für Delay wurde 60 Sekunden gewählt, dieses reicht aus um normale Fehler zu erkennen, extremere zeitliche Ausprägungen können durch eine Analyse der Laufzeit des gesamten Testsetups erkannt werde. Außerdem wird im Normalfall bei einem höherem Delay als 60 Sekunden das Netzwerk-Zeitlimit überschritten werden. Dies ist ebenfalls extern ohne Test-System erkennbar.

Ein in diesem Szenario sinnvoller Zeitstempel muss somit einen Wertebereich von 0-60000 Millisekunden abdecken können. Für die Einbettung des Zeitstempels in das Wasserzeichen werden daher 16Bit benötigt ($2^{16} = 65536$).

6.3.2 Bildidentifikations-Informationen

Bei Paketverlust kann es abhängig davon wie die Daten codiert sind dazu kommen, dass einzelne Bilder im Videostrom fehlen oder dass der Bildinhalt verändert wird.
Sollte ein Bild fehlen, ist dies unter Umständen nicht anhand des Zeitstempels erkennbar, da nicht immer davon ausgegangen werden kann, dass die Bilder in gleichem Abstand zueinander eintreffen. Daher wird in dieser Arbeit ein Identifikationsmerkmal in jedes Bild eingebettet, welches jedes Bild eindeutig identifiziert und zusätzlich eine eindeutige Reihenfolge nachvollziehen lässt. Der eingebettete Wert, der zur Identifikation verwendet wird, ist dabei ein Zahlenwert, der mit jedem Bild hochgezählt wird, beginnend mit null. Sollte ein Wert fehlen, kann dies nachvollzogen werden, indem überprüft wird, ob die Identifikationsmerkmale der eingegangenen Bilder kontinuierlich hochzählen. Durch den Einsatz einer solchen Technik lassen sich darüber hinaus auch noch andere Fehler, wie Einfrieren des Bildes und falsche Reihenfolge der Bilder, erkennen. Dies ist jedoch nicht der Fokus dieser Arbeit und wird daher nicht weiter untersucht.

Eine eindeutige Bildidentifikation muss ebenfalls lediglich einen sinnvollen Zeitraum bzw. eine sinnvolle Anzahl an Bildern abdecken. Bei Videoübertragungen sind 25-60 Bilder pro Sekunde

Standard. Um ein allgemeingültiges Wasserzeichen zu erstellen, werden 60 Bilder pro Sekunde angenommen. Bei den zeitlichen Fehlern wurde bereits ein Zeitraum von einer Minute für sinnvoll erachtet. Durch den Zähler soll ein ähnlicher Zeitraum Betrachtung finden. Bei 60 Bildern pro Sekunde ergeben sich für einen Zeitraum von ca. einer Minute ca. 3600 einzeln zu identifizierende Bildern. Um eine solche Zahl binär darstellen zu können, werden 12 Bits benötigt.

6.3.3 Fehlerkorrigierender Code

Wie im Abschnitt 6.2 bereits erklärt, basiert die Fehlererkennung des Wasserzeichens darauf, dass das Wasserzeichen durch Fehler im Bild gestört wird und sich die Werte nicht mehr korrekt extrahieren lassen. Damit würde sich der binäre Wert, der extrahiert wird, verändern. Durch den fehlerkorrigierenden Code kann das Prüfsystem solche Veränderungen einzelner Werte innerhalb der Zeichenfolge erkennen.
Im Rahmen dieser Arbeit wird der BCH-Code als fehlerkorrigierender Code eingesetzt. Der BCH-Code wurde bereits in Kapitel 2.4 vorgestellt. Der BCH-Code erlaubt es, Veränderungen an einer binären Zeichenfolge zu erkennen [2, p. 162]. Dies geschieht, indem der BCH-Code zusätzliche Informationen zur Zeichenfolge hinzufügt, die dazu verwendet werden, die Veränderungen an der Zeichenfolge zu detektieren und sogar korrigieren zu können.

Die Länge des fehlerkorrigierenden Codes ist abhängig von der gewählten maximalen Einbettungskapazität. Im Rahmen dieser Arbeit wird im Kapitel 8 eine Einbettungskapazität von mindestens 220 Bits errechnet. Dadurch ist die Anzahl der redundanten Stellen $(220 - 16) - 12 = 192$. Dies ergibt eine Fehlerkorrektureigenschaft von 30 Bits.[2] Es lassen sich also 30 Fehler durch den fehlerkorrigierenden Code korrigieren.

6.4 ZUSAMMENFASSUNG

Im vorhergehenden Kapitel wurde ein semi-fragiles Wasserzeichen vorgeschlagen, welches es erlaubt, eine Videoverarbeitungskette automatisch auf Fehler zu prüfen. Dabei wurde aufbauend auf den im Kapitel 3 vorgestellten verwandten Arbeiten eine Technik entwickelt, mit der sich zusätzliche Informationen blockweise in ein Bild einbetten lassen. Der visuelle Einfluss des Wasserzeichens wird in Kapitel 9.1.1 untersucht. Die semi-fragile Eigenschaft des Wasserzeichens in Kombination mit den eingebetteten Daten wird dazu verwendet, die in Kaptiel 5.2.2 definierten Fehler zu erkennen.

Basierend auf der Arbeit von Eugen T. Lin et al.[18] wird das Wasserzeichen in das durch die DCT transformierte Bild eingebettet, um eine höhere Robustheit gegenüber verlustbehafteter Kompression zu erhalten und den visuellen Einfluss zu verringern. Das Einbetten selbst geschieht durch das blockweise Verschieben der Summe der DCT-Koeffizienten. Basierend auf der Idee des „Spreadings“ (Streuen)[7, p. 279ff], wird die durch diese Verschiebung hervorgerufene Veränderung über die DCT-Koeffizienten gestreut. Auf diese Weise wird der einzelne DCT-Koeffizient nur leicht verändert.
Durch die Verschiebung lassen sich binäre Werte, entweder 0 oder 1, in jeden Block einbetten. Im Gegensatz zum Vorschlag von Peter Wayner[7, p. 279ff] wurde dazu der Wertebereich, durch die Verwendung mehrere Schwellwerte, in mehrere Teilabschnitte eingeteilt, wobei jeder Teilabschnitt einem der binären Werte entspricht. Ein solcher Teilabschnitt wird als Bucket bezeichnet. Welcher binäre Wert eingebettet wird, wird von dem Bucket bestimmt, in den die DCT-Koeffizienten verschoben werden. Durch die so entwickelte Erweiterung kann die Distanz

[2] Berechnet mit der durch Octave bereitgestellte Funktion: bchpoly()

um die Verschoben wird signifikant verringert werden. Bei der Extraktion wird das Wasserzeichen blockweise überprüft, indem die Summe der DCT-Koeffizienten berechnet wird. Um höhere Robustheit zu erreichen und Fehler von Kompressionsartefakten unterscheiden zu können, erfolgt die Veränderung der Informationen gewichtet.

Die Fehlererkennung wird durch die semi-fragile Eigenschaft des Wasserzeichens in Verbindung mit den eingebetteten Daten realisiert. Dabei werden folgende Informationen binär eingebettet:

- Zeitliche Informationen
- Bildidentifikations-Informationen
- Fehlerkorrigierender Code

Als zeitliche Informationen wird ein Zeitstempel eingebettet, welcher die Messung von Delay und Jitter erlaubt. Zur Detektion fehlender Bilder oder falscher Bildreihenfolgen wird die Einbettung eines Bildzählers als Bildidentifikations-Information vorgeschlagen. Der Einsatz eines fehlerkorrigierenden Codes erlaubt das Detektieren von Bildfehlern: Durch Bildfehler wird das Wasserzeichen im Bild verändert, durch den fehlerkorrigierenden Code lassen sich diese Fehler erkennen und sogar der entsprechende Block, in dem der Fehler auftritt, lokalisieren.

7 IMPLEMENTIERUNG

IMPLEMENTIERUNG

In diesem Kapitel werden einige Besonderheiten, die bei der Umsetzung des Konzepts als Prototyp aufgefallen sind, erläutert und erklärt.
Zuerst wird erklärt, wie genau der Versuchsaufbau im Rahmen dieser Arbeit vorgenommen wurde und erläutert, wie die einzelnen Tools automatisiert angesteuert werden. Danach werden ausschnitthaft einige Herausforderungen und Besonderheiten, die sich im Laufe der Implementierung der Prototypen ergeben haben, erklärt. Dabei wird erst einmal ein Überblick über das Einbettungs- sowie über das Prüfsystem gegeben. Anschließend werden die verwendeten Tools vorgestellt. Abschließend ist beschrieben, wie die in Kapitel 5.2.2 definierten Fehler simuliert werden können.

7.1 PROTOTYPEN-IMPLEMENTATION

In diesem Abschnitt soll kurz beschrieben werden wie der Versuchsaufbau während der Evaluation des Prototyps aussah. Dabei kam eine Reihe von Werkzeugen zum Einsatz.
Der Prototyp wurde in Python umgesetzt und bestand aus zwei eigenständig agierenden Modulen. Ein Modul, welches das Wasserzeichen in das Video einbettet und eines welches das Wasserzeichen anschließend wieder extrahiert. Das En- und Decodieren der Videodaten wurde mit dem x264, der H.264-Implementierung die in ffmpeg enthalten ist[34], durchgeführt.
Die einzelnen Komponenten wurden mithilfe eines Bash-Skripts automatisiert angesteuert, wobei das Bash-Skript die einzelnen Module und die Werkzeuge nacheinander ausführte. Mithilfe des Skripts wurden auch die einzelnen Parameter wie die Gewichtung und der Schwellwert definiert und bei Programmstart der einzelnen Module als Kommandozeilenargument übergeben.
Wenn ein Fehler detektiert wurde, wird die Position des entsprechenden Blocks in eine externe Textdatei geschrieben. Nach Durchlaufen der Extraktion wird die Anzahl der erkannten Fehler auf der Konsole ausgegeben und ebenfalls in eine gesonderte Textdatei geschrieben.

Um die im Konzept beschriebene DCT durchzuführen, werden die DCT- und inverse DCT-Implementierung der SciPy-Bibliothek verwendet[29, 30]. Das Encodieren der Daten wird dabei mit dem x264 [34] durchgeführt, wobei die im .YUV-Dateiformat[1] vorliegenden Daten mit dem x264 und einem passenden QP encodiert werden und anschließend mit ffmpeg wieder zurück ins .YUV-Dateiformat überführt werden.

7.1.1 Lesen der Daten

Wie im Kapitel 6 beschrieben ist vorgesehen, dass der Algorithmus die Daten aus einer Video-Quelle erhält, im Rahmen eines Video-Konferenzsystem wäre dies typischerweise eine Kamera. Darauf wird im Rahmen der Umsetzung der Prototypen jedoch verzichtet, da Livedaten von einer Kamera in der Regel schwer reproduzierbar sind. Und sich die Ergebnisse der Arbeit nicht einfach durch Dritte reproduzieren ließen.

Stattdessen werden standardisierte Videos aus öffentlichen Datenbanken verwendet[26] die direkt von der Festplatte geladen werden. Die Videos liegen hier im unkomprimiertem Y4M-Format vor. Mithilfe von ffmpeg wurden diese Daten in das .YUV-Dateiformat umgewandelt, welches ein Dateiformat ist, in dem die Bilder unkomprimiert vorliegen.

[1] in .YUV-Dateien sind YUV-Bilder unkomprimiert gespeichert. Dabei liegen die Bilder unkomprimiert und ohne zusätzliche Metadaten hintereinander auf der Festplatte.

Das grundlegende Vorgehen ist dabei folgendermaßen:

1. Laden der Daten von der Festplatte
2. Einbettung des Wasserzeichens
3. Schreiben der Daten auf die Festplatte
4. Encodieren der Daten
5. Decodieren der Daten
6. Lade Daten von der Festplatte
7. Extraktion des Wasserzeichens
8. Überprüfen des Wasserzeichens

Für die Prototypen wurde das Wasserzeichen lediglich in den Luminanz-Kanal eingebettet, die anderen Kanäle wurden beim Einbetten und Extrahieren ignoriert. Denkbar ist der Einsatz des Wasserzeichens jedoch auch auf den beiden anderen Kanälen. Dies könnte einen interessanteren Ansatzpunkt für nachfolgende Forschungen bieten, so könnte durch den Einsatz auf mehreren Kanälen eventuell eine höhere Fehlerkennungsrate erreicht werden.

Im verwendeten Versuchsaufbau findet das Einbetten und Extrahieren der Daten nicht parallel statt, sondern nacheinander. Es werden also erst alle Bilder mit einem Wasserzeichen versehen, anschließend wird das so entstandene Video mit dem x264 encodiert. Danach wird das Wasserzeichen aus allen Bildern nacheinander extrahiert. Diese Vorgehensweise ändert jedoch am Algorithmus nichts und wird lediglich verwendet, da sich die Daten durch Zwischenspeichern besser analysieren lassen. Der eigentliche Algorithmus der Prototypen ist dabei durchaus in der Lage bildweise zu arbeiten und tut dieses intern auch, das beschrieben Vorgehen wird lediglich durch eine vorangehende Schleife erzwungen. Dabei werden die einzelnen Bildern von der Festplatte aus unkomprimierten „.YUV-Datei“ gelesen und in eben diese wieder geschrieben. Ffmpeg codiert und encodiert diese anschließend mit dem x264. Die Dateien werden danach wieder mithilfe von ffmpeg zurück „.YUV-Dateiformat“ überführt, woraufhin sie durch das Prüfsystem gelesen wird und das Wasserzeichen anschließend extrahiert wird.

Folgend wird ausschnitthaft der Vorgang des Ladens von der Festplatte kurz beschrieben. Ein Auszug des Quellcodes ist im Codebeispiel 7.1 dargestellt.

```
1  def loadByteStream(self, filename, frameNr, width, height, blockSize):
2        self.Image =np.zeros((height,width))
3        bytesToRead = width * height * 3/2
4        bytesToSkip = frameNr * bytesToRead
5        file = open(filename, 'rb')
6        try:
7            file.seek(int(bytesToSkip))
8            buf = file.read(int(bytesToRead))
9        finally:
10           file.close()
11       self.blocks = []
12       for y in range(int(height/blockSize)):
13           for x in range(int(width/blockSize)):
14               list = self.getBlock(buf, x, y, width, height, blockSize)
15               reshapedBlock =np.reshape(np.asarray(list),[blockSize,blockSize])
16               self.blocks.append(reshapedBlock)
```

Codebeispiel 7.1: Das Laden eines einzelnen Bildes von der Festplatte

In dem Ausschnitt aus dem Programmcode 7.1 ist die Funktion „loadByteStream“ dargestellt, die dafür zuständig ist, die Daten von der Festplatte zu laden. Das Video wird dabei bildweise

geladen und dem Algorithmus auch bildweise zur Verfügung gestellt, so wie es in einem realen Szenario ebenfalls der Fall wäre. Direkt zum Anfang der Funktion wird dabei Speicher für die Bilddaten in Form eines zweidimensionalen Arrays allokiert.

Anschließend wird die Datei mit der in Python eingebauten „open"-Funktion als Binärdatei, gelesen. Dabei werden beim Lesevorgang alle nicht benötigten Bilder übersprungen und das aktuelle Bild gelesen. Aufgrund der bildweisen Bearbeitung durch den Algorithmus werden die Daten hier auch bildweise geladen. Dafür wird die Nummer des zu lesenden Bildes an die Funktion übergeben, mithilfe der Nummer werden die zu überspringenden Bytes errechnen und die nachfolgenden Bytes werden gelesen.

Die Datei wird anschließend geschlossen und das Bild wird in Blöcke aufgeteilt, deren Größe, durch einen Funktionsparameter variiert werden kann. Die so erhaltenen Blöcke werden in einer Member-Variable gespeichert, wo sie dem Algorithmus jederzeit zur Verfügung steht.

Analog werden die Bilder auch wieder bildweise geschrieben. Die Ausgabedatei wird nach der Verarbeitung jeden Bildes geöffnet und das Bild wird ans Ende der Datei geschrieben. Zu beachten ist, dass das Bild in Blöcke aufgeteilt wird, wobei die Blöcke eine feste Blockgröße haben. Es ist jedoch nicht möglich beliebige Auflösungen durch beliebige Blockgrößen restlos einzuteilen. Daher wird im Rahmen der Prototypen lediglich die Teile des Bildes betrachtet, die sich Restlos durch die verwendete Blockgröße teilen lassen. Diese Entscheidung musste aufgrund der zeitlichen Einschränkung der Arbeit getroffen werden. Für zukünftige Untersuchungen könnte interessant sein, wie sich der Prototyp erweitern lässt, um auch die übrigen Regionen des Bildes prüfen zu können. Einige Ideen wie sich diese Herausforderung lösen ließe, sind im Kapitel 10.3 vorgestellt.

7.1.2 Der erweiterte Algorithmus

In Kapitel 6 wurde der Algorithmus zum Einbetten des Wasserzeichens vorgestellt, der Algorithmus geht dabei wie folgt vor:

1. DCT-Transformation des Blocks
2. Ermittlung des Wertes Target
3. Summierung der Koeffizienten
4. Berechnung der Differenz zwischen Target und der Summe der Koeffizienten
5. „Verschieben" der Koeffizienten
6. Inverse Transformation des Blocks

Dieses Vorgehen ließ sich jedoch nicht genauso implementieren und musste daher angepasst werden. Die Anpassung und ihrer Gründe werde im folgendem Abschnitt erläutert.

Fließkomma Transformation

Das .YUV-Dateiformat reserviert für einen Pixel im Luminanz-Kanal einen Byte, also 8 Bits. Mit diesen 8 Bits lassen sich Ganzzahlwerte zwischen 0 - 255 darstellen.
Die DCT-Transformation und inverse Transformation ist jedoch nicht auf die Arbeit von Ganzzahlwerten begrenzt. Bei einer inversen DCT-Transformation kann es so vorkommen, dass

Fließkommazahlen als Ergebnis entstehen. Diese Fließkommazahlen lassen sich im .YUV-Dateiformat, welches 8 Bit pro Kanal pro Pixel definiert, nicht darstellen. Sie werden also spätestens beim Schreiben der Daten in besagtes Dateiformat, umgewandelt. Diese Umwandlung kann zu einem signifikanten Informationsverlust führen.
Zur Illustration wird nachfolgend ein Rechenbeispiel durchgegangen. Es sei ein 2×2 großer Block gegeben, der komplett aus schwarzen Pixeln besteht. Damit entsprechen alle Pixelwerte den Wert 0. Durch eine DCT-Transformation wird dieser Block nun transformiert. Als Ergebnis ergibt sich eine Matrix, die ebenfalls aus Nullen besteht. Angenommen, dieser Block soll mit dem in Kapitel 6 vorgestelltem Wasserzeichen versehen und der gewünschte Target-Wert liegt bei eins, dann liegt, nach dem Verschieben der Summe der DCT-Koeffizienten, der DC-Wert ebenfalls bei eins[2]. Der Vorgang ist in Gleichung 7.1 illustriert. Wie sich zeigt, sind die Werte nun Fließkommazahlen und damit im .YUV-Format nicht mehr darstellbar. Beim Schreiben würden sie in einen Ganzzahlwert umgewandelt.

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \overset{\text{DCT}}{\Rightarrow} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \overset{\text{Verschieben}}{\Rightarrow} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \overset{\text{iDCT}}{\Rightarrow} \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \tag{7.1}$$

Ein solcher „Rundungsfehler“ kann einen signifikanten Informationsverlust zur Folge haben. So entspricht eine Rundung aller Koeffizienten um 0.5 bei einem 64×64 Pixel großem Block einer Veränderung der Summe um 2048.

Als Lösung dieser Herausforderung wird im Prototypen ein iterativer Prozess gewählt. Hierbei wird der Block verschoben, anschließend wird nach der Inversen DCT Transformation und der Typenumwandlung erneut untersucht, ob die angestrebte Verschiebung auch nach dem Abschneiden der Nachkommazahl noch gegeben ist. Sollte durch die Rundung erneut eine Differenz zwischen dem Zielwert „Target“ und der Summe der DCT-Koeffizienten entstanden sein, wird erneut die Differenz errechnet und um diese neue Differenz verschoben. Dafür wurden diese Schritte - DCT-Transformation, Verschiebung der Summe des Blocks und inverse DCT-Transformation, Rundung und erneute DCT-transformation - in eine Schleife eingebettet. Durch die inverse Transformation mit anschließender Rundung und erneuter Transformation kann erneut die Summe berechnet werden. Sollte es zum Informationsverlust gekommen sein, so kann erneut verschoben werden. Durch die Schleife nähert sich der Block immer weiter dem Zielwert an. Die Schleife terminiert, wenn die Summe der DCT-Koeffizienten den Zielwert erreicht haben.

Maximalwert

Als weitere Herausforderung hat sich herausgestellt, dass durch verschieben und anschließender inverser DCT, Werte erzeugen können die oberhalb des durch ein Byte darstellbaren Bereichs liegen. Bei der Rücktransformation durch die inverse DCT von vorher verschobenen DCT-Koeffizienten, kann es vorkommen, das Pixelwerte erreicht werden, die über bzw. unter dem erlaubten Wertebereich von 0-255 liegt.

Analog zu dem Problem aus Abschnitt 7.1.2 ist dies mit einem Rechenbeispiel angegeben. Das Rechenbeispiel ist in Gleichung 7.2 dargestellt. Hier ist ein weißer 2×2 Pixel großer Block gegeben, weiß bedeutet in diesem Fall, dass jeder Eintrag der Matrix, dem Wert 255 entspricht. Der Zielwert ist für dieses Beispiel auf 512 festgelegt.

$$\begin{bmatrix} 255 & 255 \\ 255 & 255 \end{bmatrix} \overset{\text{DCT}}{\Rightarrow} \begin{bmatrix} 510 & 0 \\ 0 & 0 \end{bmatrix} \overset{\text{Verschieben}}{\Rightarrow} \begin{bmatrix} 512 & 0 \\ 0 & 0 \end{bmatrix} \overset{\text{iDCT}}{\Rightarrow} \begin{bmatrix} 256 & 256 \\ 256 & 256 \end{bmatrix} \tag{7.2}$$

[2] Die Veränderung wurde hierbei zur besseren Illustration auf den DC-Wert beschränkt.

Durch die Veränderung entstehen Pixelwerte mit dem Wert 256. Im .YUV-Datenformat werden für das Speichern eines Pixels und einem Farbkanal jedoch lediglich 8 Bit verwendet. Somit lassen sich maximal ganzzahlwerte bis 255 speichern. Damit würde auch hier Informationen verloren gehen. Um auch diese Herausforderung zu lösen, wurde die in Abschnitt 7.1.2 vorgestellt Schleife erweitert. Dabei werden nach der Typenumwandlung noch alle Werte oberhalb von 255 und alle Werte unterhalb von 0 abgeschnitten. Anschließend wird überprüft, ob die Summe weiterhin dem Zielwert entspricht. Sollte dies nicht der Fall sein, wird die Schleife erneut ausgeführt, damit wird der Block erneut transformiert und verschoben. So nähert sich der Block immer weiter dem Zielwert an.

Überschwingen

Als weitere Herausforderung hat sich ergeben, dass es, durch die oben vorgeschlagene iterative Veränderung der Werte, dazu kommen kann, dass der Algorithmus in einer Endlosschleife landet. Durch die Veränderung, die durch die Typenumwandlung und das Begrenzen der Werte entsteht, kann es dazu kommen, dass die Werte den erwarteten Zielwert nicht erreichen, sondern um den zu erreichenden Wert „schwingen". Zur Illustration kann angenommen werden, dass die Summe der DCT-Koeffizienten eine Distanz d zum Target hat. Anschließend wird die Summe auf den Wert des Targets verschoben, damit ist die Distanz null. Nach anschließender Transformation, Typenumwandlung und Abschneiden zu hoher bzw. zu niedriger Werte, ist die Distanz $-d$, dabei sei d ein beliebiger Wert ungleich 0.
Um dies zu verhindern, wurde die Verschiebung iterativ vorgenommen. Die Verschiebung um eine gegebene Differenz d wurde also nicht in einer großen Verschiebung realisiert, sondern wurde annäherungsweise in Schritten durgeführt, wobei die Schrittgröße von einem Parameter β bestimmt wird. Somit nähern sich die Summen langsam dem gewollten Wert an und ein starkes Überschwingen kann verhindert werden.

Der finale Algorithmus

Daraus ergibt sich die finale Implementierung des Algorithmus. Zusätzlich zum Konzept kommt also ein iterativer Ansatz hinzu, der die Rundungsfehler durch eine iterative Annäherung und das Abklemmen der Werte auf den Wertebereich zwischen 0 und 255 abfedern soll. Daraus ergibt sich die folgende Veränderung des Algorithmus. Änderungen im Algorithmus sind kursiv dargestellt. Die unveränderten Teile des Algorithmus sind fettgedruckt dargestellt.

1. **DC-Transformation des Blocks**
2. **Ermittlung des Wertes Targets**
3. **Summierung der Koeffizienten**
4. **Berechnung der Distanz zwischen Target und der Summe der Koeffizienten**
5. **„Verschieben" der Koeffizienten** *gewichtet mit einem Wert β*
6. **Inverse Transformation des Blocks**
7. *Abklemmen aller Werte >255 & <0*
8. *Typenumwandlung der Fließkommazahlen in Ganzzahlen*
9. *Wenn: Distanz zwischen der Summe der DCT-Koeffizienten und dem Target gleich 0: Ende, Andernfalls: springe zu 1.*

7.1.3 Werkzeuge

Im Rahmen dieser Arbeit sind einige Werkzeuge und Programme zum Einsatz gekommen, um die unterschiedlichen Aufgaben zu erfüllen. Diese Tools, inklusive ihrer Versionen, sofern vorhanden, sind in Tabelle 7.1 aufgezeigt und beschrieben.

Name	Version	Beschreibung
Python	Python 3.6.1 - Anaconda custom 64 – *bit*	Python wurde als Programmiersprache des Prototypen verwendet. Aus der Anaconda-Distribution wurde die DCT und iDCT implementierung von scipy verwendet. So auch die Implementierung von numpy.
ffmpeg	3.2.4	ffmpeg wurde für das Encodieren und Decodieren sowie für einige Umwandlung zwischen unterschiedlichen Video formaten verwendet. Als Encoder ist die Implementierung des H.264-Standard zum Einsatz gekommen.
Mathplotlib	2.0.2	Mathplotlib wurde dazu verwendet, die in dieser Arbeit enthaltenen zweidimensionale sowie dreidimensionalen Plots zu erstellen.
Git-Bash	git version: 2.13.0.windows.1 GNU bash: version 4.4.12	Die Bash-Implementierung der Git-Bash wurde dazu verwendet die unterschiedlichen Programme automatisiert nacheinander zu starten.

Tabelle 7.1: Die verwendeten Tools

7.2 FEHLERPRODUZIERENDE SKRIPTE

Im Rahmen dieser Arbeit wurde ein verteiltes Video-Konferenzsystem lediglich simuliert, es fand kein tatsächlicher Eingriff in den Encoder oder Decoder statt, um die Fehler zu produzieren. Im Rahmen dieser Arbeit werden die Fehler also reproduzierbar simuliert.

Dabei wurde sich auf die in Kapitel 6 definierten Fehler konzentriert. Im folgenden Abschnitt wird beschrieben, wie die einzelnen Fehler simuliert werden. Dafür wurden für jeden einzelnen Fehler eigene Python-Skripte erstellt.

7.2.1 Bildfehler

Unter Bildfehlern werden im Rahmen dieser Arbeit solche Fehler zusammengefasst, welche sich direkt auf Bildebene auswirken. In der Verarbeitungskette können solche Fehler in allen Bereichen auftreten, sowohl durch Fehler bei der Encodierung, als auch durch Manipulationen bei der Übertragung und nicht zuletzt beim Decodieren. Im Rahmen dieser Arbeit werden diese Fehler nur simuliert und werden nach dem En- und Decodierungsvorgang direkt auf Pixelebene eingebettet. Es werden also Pixelwerte vom Algorithmus übermalt.

Blockfehler

Blockfehler sind in diesem Zusammenhang Fehler bei denen blockförmige Pixelregionen unterschiedlicher Größe auf den Wert null gesetzt werden, was im Falle von Schwarz-Weiß-Bildern der Farbe schwarz entspricht. Diese Fehler würden in einem realexistierenden Szenario natürlich innerhalb des Encoders oder Decoders eingeführt werden. Im Rahmen dieser Arbeit werden die Fehler nur simuliert, ein Eingriff in den En- oder Decoder findet nicht statt.

Der Fehler, dass sich schwarze Blöcke im Bild bilden, wird hier mithilfe eines Python Skripts simuliert. Dabei werden die Bilddaten verändert, nachdem sie en- und decodiert wurden. Dies geschieht indem, die Pixel einfarbig, in diesem Fall schwarz, übermalt werden. Dies erlaubt auch eine hohe Konfigurierbarkeit der Größe und die Position der Fehler. Der Pseudocode für das Skript, welches solche Blöcke produziert, kann in Codebeispiel 7.2 begutachtet werden, wobei *X* und *Y* die Position des Blocks angeben und *width* und *height* die Größe des Blockfehlers bestimmt.

```
def blockError(self, frame,  X, Y, width, height):
for x in range(X, width):
    for y in range(Y, height):
        frame[x,y]  = 0
```

Codebeispiel 7.2: Das Fehlerskript welches Blockfehler produziert

Im Abschnitt 9.2.1 wurde eine systematische Einbettung des Wasserzeichens benötigt. Dabei sollten die Ergebnisse zufällig sein um ein möglichst realistisches Testszenario zu erreichen, gleichzeitig sollten die Daten reproduzierbar bleiben. Welcher Block abhängig von einem gegebenen Bild von einem Blockfehler betroffen ist. Die Bestimmung einer solchen zufälligen aber reproduzierbaren Position wird in Codebeispiel 7.3 illustriert.

```
np.random.seed(frameNr)
blockNumber = np.random.randint(numberOfBlocksPerFrame)
X = (blockNumber) % int((imageWidth/blockSize))
Y = int((blockNumber) / int(ImageWidth/blockSize))
      np.random.seed(frameNr)
      randomPosX = np.random.randint(minPosX,maxPosX)
      randomPosY = np.random.randint(minPosY,minPosY)
      Xcoord = X*blockSize+blockPosX
      Ycoord = Y* blockSize + blockPosY
blockError(self, frame, X, Y, width, height)
```

Codebeispiel 7.3: Das Skript, welches die Position eines Blockfehlers bestimmt

Croppingfehler

Als Croppingfehler werden im Rahmen dieser Arbeit solche Fehler bezeichnet, welche zu einem einfarbigen Balken am Rand führen. Dies ist in Kapitel 5.2.2 noch einmal detailliert erläutert. Zur

Simulation eines solchen Fehlers geht das Skript ähnlich vor, wie bei den Bildfehlern in Abschnitt 7.2.1. Es werden ebenfalls Pixelwerte auf null gesetzt. Um die schwarzen Bereiche am Bildrand zu generieren, muss lediglich die Position des Fehlers angeben. In Codebeispiel 7.4 ist dabei der entsprechende Code gezeigt, der die Croppingfehler an der linken Seite des Bildes einfügt.

```
1  for y in range(0,Frame.width):
2    for x in range(0, thickness):
3      frame[x,y]  = 0
```

Codebeispiel 7.4: Das Skript welches die Croppingfehler einfügt. Hier ein Beispiel für den linken Rand.

7.2.2 Zeitliche Fehler

Unter zeitlichen Fehler werden in dieser Arbeit alle Fehler zusammengefasst, die das zeitliche Verhalten der Einzelbilder unerwartet verändern. Unerwartet heißt in diesem Kontext, dass die Bilder in einer anderen Reihenfolge oder zeitlichen Abfolge beim Empfänger ankommen, als sie aus der Videoquelle gekommen sind. In diesem Abschnitt wird erklärt, wie diese Fehler künstlich erzeugt werden. Dabei wird sich, wie in Kapitel 5.2.2 definiert, auf die Fehler Jitter und Delay konzentriert.

Jitter

Jitter-Fehler sind zeitliche Fehler, bei denen sich der Zeitabstand zwischen den einzelnen Bildern während der Verarbeitung verändert. Eine genaue Definition dieser Fehler ist in Kapitel 5.2.2 zu finden. Jitter-Fehler wurden im Rahmen dieser Arbeit simuliert, indem dem Extraktionsmodul ein veränderter Zeitstempel geliefert wird. Dabei wurden stark variierende Einbettungszeitpunkte simuliert. Beim Einbetten wurde der reguläre Zeitstempel mit Abständen regulärer Größe verwendet. Durch die Berechnung der Differenzen zwischen Einbettungs- und Extraktionszeitpunkt kann das Delay eines jeden Bildes einzeln berechnet werden. Darauffolgend kann das Delay der einzelnen Bilder mit denen der anderen verglichen werden. Variieren diese stark bedeutet dies das Jitter aufgetreten ist.

Delay

Delay bedeutet im Rahmen dieser Arbeit, dass eine erhebliche Differenz zwischen Einbettungszeitpunkt und Extraktionszeitpunkt existiert. Dieser wurde mithilfe eines veränderten Zeitstempels erreicht. Hierbei wurde dem Modul, welches das Wasserzeichen einbettet, ein in der Vergangenheit liegender Zeitstempel zur Verfügung gestellt. Das Prüfsystem erhält den regulären Zeitstempel. Somit wird dem Prüfsystem vorgespielt, dass der Zeitstempel bereits vor einiger Zeit die Eingebettet wurde. Das System wird durch die Errechnung der Differenz zwischen Einbettungszeitpunkt und Extraktionszeitpunkt nun ein Delay erkennen.

7.2.3 Bitstromfehler

Bei Bitstromfehlern werden solche Fehler imitiert, die durch Paketverlust verursacht werden. Das Erzeugen eines reproduzierbaren fehlerhaften Übertragungskanals ist schwierig. Im Rahmen dieser Arbeit werden die Übertragungsfehler simuliert, indem die auf der Festplatte

liegenden Daten in ein Format konvertiert werden, welches den einfachen Zugriff auf einzelne Teile des Videos ermöglicht. Anschließend wird diese Datei so verändert, wie es ähnlich auch bei Paketverlust geschehen könnte.
Hierfür wird das Video in das „Annex B"-Format konvertiert. Das „Annex B"-Format ist ein Format, wo die Daten des H.264 in aufeinanderfolgenden „Network Abstraction Layer (NAL)-Units" gespeichert sind. Eine solche NAL-Unit kann dabei als eine Art Netzwerkpaket angesehen werden. Eine NAL-Unit kann unterschiedliche Informationen beinhalten, dazu gehören unter anderem Kontrollinformationen, „Key-Frames", „P-Frames", „B-Frames" oder Slices eben dieser. Eine vollständige Liste kann im Buch „The H.264 Advanced Video Compression Standard", von Iain E. Richards gefunden werden[3, p.102]. In dem hier vorgestellten Ansatz wird lediglich ein einzelner „Key-Frame" erzeugt und darauffolgend nur noch sich darauf beziehende „P-Frames". Da Key-Frames i.d.R. größer als darauffolgende P-Frames sind, weshalb im Bereich Echtzeit-Video-Kommunikation der Einsatz von Key-Frames häufig minimiert wird.
Mit ffmpeg wird eine mit H.264 kodierte Datei verlustlos neu im Annex B-Format paketiert:

```
1
2 ffmpeg.exe  -y -i input.mp4 -c:v libx264  -x264-params "keyint=9999:min-keyint=9999:
    scenecut=-1" -bsf:v h264_mp4toannexb output.264
```

Codebeispiel 7.5: Der Befehl, um eine .mp4 Datei ins Annex B-Format umzuwandeln

Das „Annex B"-Format ist hierbei ein Format bei dem die NAL-Units direkt hintereinander geschrieben werden und durch eine Folge von 2 bzw. 3 0x00-Bytes und ein 0x01-Byte getrennt werden. Das folgende Listing zeigt die Funktion, die das, „Annex-B"-Format lädt und anschließend die Daten in einzelne NAL-Units aufspaltet.

```
1 def start(self, pathToAnnexBFile):
2   streamFileHandler = open(pathToAnnexBFile,'rb')
3         fileContent = bytearray(streamFileHandler.read(-1))
4         streamFileHandler.close()
5         self.splittedContent = fileContent.split(b"\x00\x00\x00\x01")
```

Codebeispiel 7.6: Ein Ausschnitt aus dem Programm der die „Annex B"-Datei lädt und aufspaltet

Die Daten werden dabei von der Festplatte mit der von Python bereitgestellten „*open*"-Funktion geladen. Anschließend werden die Daten zu einem Bytearray gewandelt und anschließend an der binären Folge 0*x*000001 gespalten.

Der Paket Verlust wird nun durch das Löschen von einzelnen oder aufeinanderfolgenden NAL-Units realisiert. Dies kann potentiell alle Arten von NAL-Units treffen, also nicht zwangsläufig nur solche die Bildinformationen beinhalten. Da dies einen zufälligen Paketverlust simulieren soll, ist dies jedoch ähnlich in realem Szenario denkbar. Das folgende Listing zeigt den Code der einzelne NAL-Units aus der vorliegenden Datei löscht:

```
1   def deleteFrames(self, positionsToDelete, splittedContent):
2         for j in reversed(positionsToDelete):
3             del splittedContent[j]
4         return splittedContent
```

Codebeispiel 7.7: Das Löschen von einzelnen NAL-Units

Wobei der Array *positionsToDelete* die zu löschende NAL-Unit enthält. In Abschnitt 9.2.3 werden die Ergebnisse dieser Untersuchungen dargestellt. Dabei wurden systematisch von vorne nach hinten durchgehend, die einzelnen NAL-Units gelöscht.

7.3 ZUSAMMENFASSUNG

In diesem Abschnitt wurden die Besonderheiten der Implementierung beleuchtet. Dabei wurde erst beschrieben wie der Versuchsaufbau aussieht. Anschließend wurden allgemeine Besonderheiten beim Ablauf erläutert. Darauffolgend wurde der im Konzept entwickelte Algorithmus um einige Mechanismen erweitert. Anschließend wurden kurz die verwendeten Werkzeuge vorgestellt. Abschließend wurde erklärt, wie die Fehler simuliert werden.

Die prototypische Umsetzung besteht aus 2 getrennten Modulen. Eines der Module ist für das Einbetten der Daten zuständig. Das andere Modul ist für das Extrahieren und Überprüfen der Daten zuständig. Diese beiden Systeme werden automatisiert durch ein Bash-Skript angesprochen, welches auch die Kompression die durch den x264-Encoders von ffmpeg parametrisiert und ausführt.

Die Implementierung konnte nicht genauso umgesetzt werden, wie es im Konzept vorgestellt wurde, da durch die Transformationen sowohl Gleitkommazahlen als auch Werte höher als 255 entstehen können, die im Videoformat nicht abbildbar waren. Daher wurde der Algorithmus um eine Schleife erweitert, der sich iterativ dem gewünschten Wert annähert und nach jedem Iterationsschritt die ungültigen Werte korrigiert. Durch diese Korrekturen kann es jedoch sein, dass sich Summe sich kontinuierlich verändert, sodass die gewünschte Verschiebung nie erreicht wird. Daher wurde der im Kapitel 6 entwickelte Algorithmus erweitert.
Nachfolgend ist der erweiterte Algorithmus noch einmal dargestellt. Erweiterungen am Algorithmus sind dabei kursiv dargestellt. Die unveränderten Teile des Algorithmus sind fettgedruckt dargestellt.

1. **DC-Transformation des Blocks**
2. **Ermittlung des Wertes „Targets“**
3. **Summierung der Koeffizienten**
4. **Berechnung der Distanz zwischen Target und der Summe der Koeffizienten**
5. **„Verschieben“ der Koeffizienten**
6. **Inverse Transformation des Blocks** *gewichtet mit einem Wert β*
7. *Abklemmen aller Werte >255*
8. *Typenumwandlung der Fließkommazahlen in Ganzzahlen*
9. *Wenn: Distanz zwischen der Summe der DCT-Koeffizienten und dem Target gleich 0: Ende, Andernfalls: springe zu 1.*

Die drei Fehlerkategorien, in Kapitel 5.2.2 definierten Fehler, wurden durch unterschiedliche Vorgehen simuliert:

Bildinhaltsfehler: Die Bildinhaltsfehler wurden simuliert indem die zu überprüfenden Fehler direkt in das Bild eingebettet wurden.

Zeitliche Fehler: Die zeitlichen Fehler wurden dadurch generiert, dass die vom System gelieferten Zeitstempel variiert wurden.

Bitstromfehler: Um Paket Verlust zu simulieren wurde das Video in das „Annex B“-Format überführt. Durch Löschen einzelner NAL-Units und dem Rücktransformieren ließen sich so Bitstromfehler simulieren.

8 ANALYSE

ANALYSE

Im folgenden Kapitel wird die Umsetzung des Konzepts und der Einfluss der unterschiedlichen Parameter auf die Eigenschaften des Wasserzeichens untersucht. Die Detektion des Wasserzeichens soll dabei möglichst robust bezüglich der Kompressionseffekte des Video-Encoders sein und gleichzeitig eine zuverlässige Erkennung der in Abschnitt 5.2.2 beschriebenen Fehlerbilder erlauben. Zusätzlich soll der visuelle Einfluss, der durch das Wasserzeichen entsteht, minimiert werden.

Das folgende Kapitel ist wie folgt aufgebaut: Zuerst wird analysiert, welchen Einfluss verschiedene Parameter wie Gewichtung und Schwellwert auf die Robustheit haben. Anschließend wird die Fehlererkennung in Abhängigkeit derselben Parameter untersucht. Als nächstes wird evaluiert, welche Parameter die Anforderungen, die in Kapitel 5 definiert wurden, am besten erfüllen. Anschließend wird eine kurze Analyse des visuellen Einflusses vorgenommen. Aufbauend auf diesen Analysen werden Parameter gewählt, die die Anforderungen am besten erfüllen. Die so erarbeiteten Parameter bilden die Grundlage für die Auswertung in Kapitel 9.

Um die Ergebnisse nachvollziehbar und überprüfbar zu halten, wurden Videos aus der öffentlich zugänglichen „Xiph"-Datenbank[26] verwendet. Da diese Arbeit im Rahmen eines Videokonferenzsystems entsteht, wurden Videos gewählt, die einer Videokonferenz-Situation ähneln. Folgende Videos wurden ausgewählt: Akiyo, Crew, Deadline, Mother_daughter, News, Paris[26]. In Abbildung 8.1 ist zur Illustration jeweils eine Momentaufnahme aus jedem Video dargestellt.

Abbildung 8.1: Einzelbilder der verwendeten Testsequenzen (von links oben nach rechts unten): Akiyo, Crew, Deadline, Mother_daughter, News, Paris

Fokus dieser Arbeit ist die Auswahl geeigneter Techniken und die prototypische Umsetzung des in der Aufgabenstellung beschriebenen Testsystems. Performance-Aspekte der Algorithmen wurden im Rahmen dieser Arbeit nicht betrachtet, da diese im realen Anwendungsfall sehr stark von der Implementierung abhängen (z.B. hardware-beschleunigte Videoverarbeitung, verfügbarer Festplatten- und Netzwerkdurchsatz).

8.1 ROBUSTHEIT

Im folgenden Abschnitt soll das Wasserzeichen auf seine Robustheit untersucht werden. Die Veränderungen, die ein verlustbehafteter Video-Encoder (z.B. nach dem H.264-Standard) an den Pixeldaten des Videos vornimmt, sind zwar visuell unauffällig, aber messtechnisch signifikant: sie betreffen somit auch das in den Pixeldaten eingebettete Wasserzeichen. Als Robustheit wird somit die Eigenschaft des Wasserzeichens bezeichnet, auch nach der verlustbehafteten Kompression weiterhin intakt zu sein. Eine solche, durch den Encoder hervorgerufene Veränderung ist beispielhaft in Abbildung 8.2 dargestellt. Die Robustheit wird daran gemessen, wie viele eingebettete Bits pro Bild noch korrekt extrahiert werden können, nachdem das Video en- und decodiert wurde.

(a) Foreman vor der Encodierung

(b) Foreman nach der Encodierung mit dem x264 und einem QP von 40

Abbildung 8.2: Ein Ausschnitt des Videos „Foreman“ vor (a) und nach dem Encodieren (b).

Als Erstes wird eine sinnvolle Blockgröße definiert. Daraufhin wird untersucht, welchen Einfluss die bereits im Konzept erwähnte Gewichtung auf die Robustheit hat. Dazu wird die durchschnittliche Größe der DCT-Koeffizienten gemittelt aus mehreren Bildern untersucht. Durch die Kenntnis der durchschnittlichen Größe der einzelnen DCT-Koeffizienten lässt sich eine proportional passende Veränderung der DCT-Koeffizienten realisieren. Abschließend wird untersucht, wie sich die Veränderungen der verlustbehafteten Kompression auf die DCT-Koeffizienten auswirken.

8.1.1 Blockgröße

Wie im Konzept erläutert, wird das Wasserzeichen blockweise eingebettet. Dabei ist die Einbettungskapazität und die Robustheit des Wasserzeichens abhängig von der gewählten Blockgröße. In diesem Abschnitt soll eine Blockgröße definiert werden, die robust ist und gleichzeitig erlaubt, die in Kapitel 5.2.2 definierten Ziele zu erreichen.

Die Robustheit der Technik basiert auf der Idee, Informationen über ein Trägermedium zu streuen[7]. In diesem Fall ist das Trägermedium ein einzelner, mit der DCT transformierter, Block. Pro Block wird ein Informationsbit aus den in Kapitel 6.3 definierten Informationen, kodiert. Das Potential, auf wie viele Koeffizienten das Wasserzeichen gestreut werden kann, hängt dabei direkt von der Blockgröße ab und hat damit voraussichtlich einen Einfluss auf die

Robustheit. Daher wird versucht, eine möglichst hohe Blockgröße zu verwenden. Im Rahmen dieser Arbeit, wird sich bei der Analyse auf Blockgrößen konzentriert, bei denen es sich um Zweierpotenzen handelt. Das heißt, Blockgrößen die durch $N^2 \mid N \in \mathbb{N}$ definiert sind.

Im Rahmen der Analyse soll eine Blockgröße definiert werden, die die Grundlage für die weitere Analyse bietet. Mit der Blockgröße variiert auch die Anzahl der DCT-Koeffizienten eines solchen Blockes. Mit der Anzahl der DCT-Koeffizienten ändert sich auch ihre Summe teilweise signifikant. Dies kann in Abbildung 8.3 nachvollzogen werden. Somit müsste mit der Veränderung der Blockgröße auch der Schwellwert angepasst werden.
Zusätzlich müsste eine Gewichtungsfunktion an die variierende Anzahl an Koeffizienten angepasst werden. Im Rahmen dieser Arbeit war es somit nicht möglich, allgemeingültige Aussagen für alle Blockgrößen zu treffen, da es zu viele sich mit der Blockgröße verändernde Variablen gibt. Spätere Forschungsarbeiten könnten sich damit befassen, wie sich die in dieser Arbeit entwickelte Technik auf andere Blockgrößen skalieren lässt.

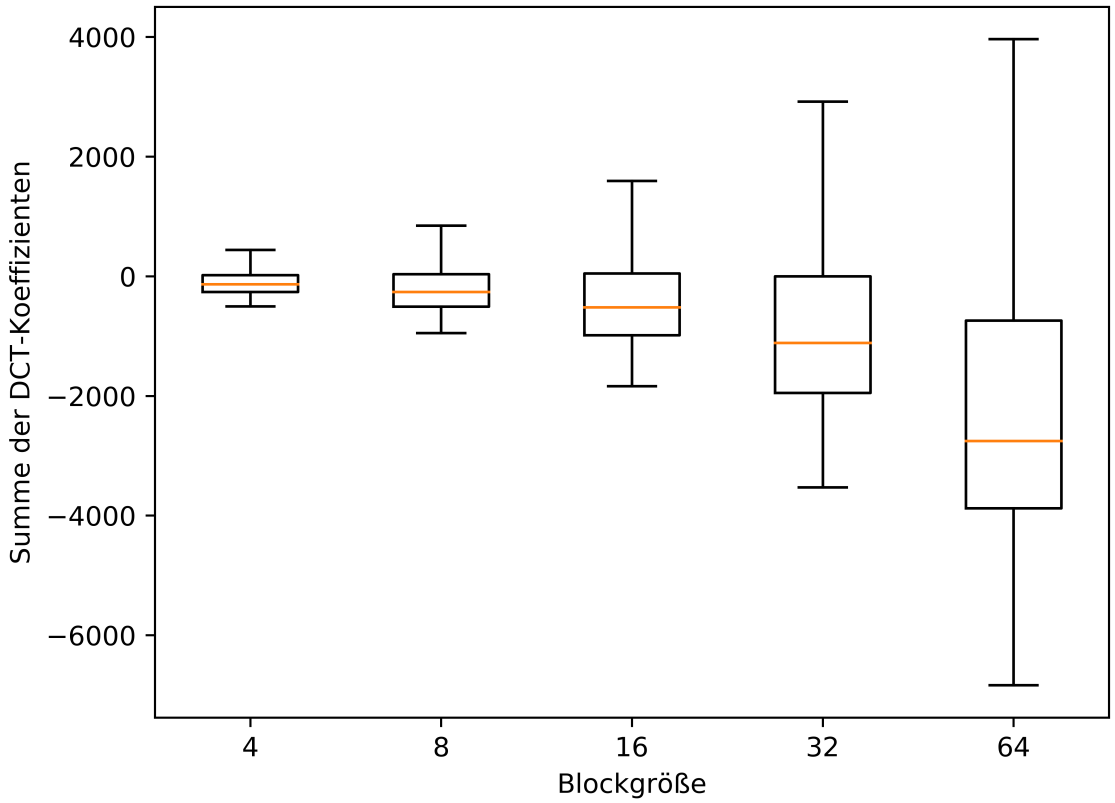


Abbildung 8.3: Die Summen der Blöcke mehrerer Videosequenzen in Abhängigkeit von der Blockgröße. Für die Darstellung wurde der Boxplot[35] verwendet. Bei der hier gegebenen Darstellung wurde auf die Darstellung der Ausreißer zum Zweck der Übersicht verzichtet.

Generell lässt sich die Anzahl an möglichen Blöcken(K), abhängig von der Auflösung, durch folgende Formel beschreiben, wobei w_{Image} und h_{Image} die Breite und Höhe des Bildes darstellen und w_{block} und h_{block} die Breite und Höhe der Blöcke angibt:

$$K = \left\lfloor \frac{w_{Image}}{w_{block}} \right\rfloor \cdot \left\lfloor \frac{h_{Image}}{h_{block}} \right\rfloor \tag{8.1}$$

Wie dem Kapitel 6.3 zu entnehmen ist, ist eine Kapazität von 28 Informationsbits zwingend

notwendig, um alle für die zeitliche Analyse notwendigen Daten einbetten zu können. Zudem ist eine Redundanz erforderlich, die es erlaubt, die in Kapitel 5.2.2 definierten Bildfehler zu erkennen. Da die Kapazität sowohl von der Blockgröße als auch von der Auflösung abhängt, wird die Blockgröße anhand der zu erkennenden Fehler und der erwarteten Auflösung definiert. Heutzutage sind Kameras mit einer Auflösung von 1280×720 quasi Standard. Selbst günstige Webcams haben häufig eine solche oder eine höhere Auflösung. Diese Auflösung wird daher als minimale Auflösung für die Arbeit festgelegt.

Sollte es notwendig sein, die hier entwickelte Technik auf kleineren Bildern anzuwenden, wäre eine Möglichkeit, die Informationen, die sonst auf ein Bild gestreut werden, über mehrere Bilder fortlaufend einzubetten. Alternativ wäre die Verwendung von kleineren Blockgrößen ebenfalls denkbar. Hierfür wäre jedoch wie bereits erwähnt noch zusätzliche Forschungsarbeit notwendig. Die Verwendung von höheren Auflösungen ist durch die Erhöhung der Redundanz möglich.

Die Definition der Blockgröße findet nachfolgend anhand der von ihr zu erkennenden Fehler statt. Dabei soll die Blockgröße möglichst hoch gewählt werden um die Streuung die in Kapitel 6.2 erklärt wurde, möglichst zu maximieren. Gleichzeitig sollen die in Kapitel 5.2.2 definierten Fehler weiterhin erkannt werden. Um dies zu ermöglichen müssen die zeitlichen Informationen und die Bildidentifikationsinformationen zwangsläufig Teil des Wasserzeichens sein. Zusätzlich müssen genug Redundanzen gegeben sein um die erwarteten Bildfehler zu erkennen.

Blöckgröße	Einbettungskapazität	Anzahl horizontale Croppingfehler	Anzahl vertikale Croppingfehler
512×512	2	2	1
256×256	10	5	2
128×128	50	10	5
64×64	220	20	11
32×32	880	40	22

Tabelle 8.1: Die Anzahl, der durch Cropping betroffenen Blöcke in Abhängigkeit der Blockgröße bei einer Auflösung von 1280×720.

Hierzu ist in Tabelle 8.1 die Anzahl der Blöcke in Abhängigkeit von der Auflösung bei unterschiedlichen Blockgrößen dargestellt. Zusätzlich wurde die Anzahl der durch Croppingfehler betroffene Blöcke dargestellt. Die durch Croppingfehler betroffenen Blöcke sind die Anzahl der horizontal bzw. vertikal in einer Reihe eingebetteten Blöcke. Die Anzahl der durch Blockfehler betroffene Blöcke wird nicht betrachtet, da die Anzahl der Blockfehler von vielen Faktoren abhängig ist. Zudem ist die Anzahl der Blockfehler potentiell höhere als die Anzahl der Blöcke. Der BCH-Code kann Fehler die höher als die Redundanz sind nicht erkennen.
Als Maßstab der minimal zu erkennenden Bildfehler werden daher die Croppingfehler verwendet. Weder eine Blockgröße von 512×512 noch von 256×256 Pixeln hat bei einer Auflösung von 1280×720 Pixeln eine Einbettungskapazität die größer als die erforderlichen 28 Blöcken sind. Eine Blockgröße von 128×128 Pixeln besitzt zwar die notwendige Einbettungskapazität von über 28 Bits, allerdings bleiben lediglich 22 Bits für die Redundanz, was lediglich die Korrektur von 3 Fehlern erlaubt. [1]. Dies ist weder genug um horizontale noch vertikale Croppingfehler zu erkennen. Damit ist eine Blockgröße von 64×64 Pixel die maximal mögliche Größe. Die Einbettungskapazität ist ausreichend um die 28 Bits einzubetten und besitzt zusätzlich eine Fehlerkorrektureigenschaft von 30 Bits.[2]

8.1.2 Gewichtung

Die im Rahmen dieser Arbeit vorgestellte Methode arbeitet mit dem x264-Encoder, die Implementierung des H.264-Standards die ffmpeg verwendet. Der H.264-Standard definiert

[1] Die Anzahl der korrigierbaren Bits wurde mithilfe der von Octave bereitgestellten Funktion „bchpoly()“ ermittelt
[2] Siehe Fußnote 1

neben der Quantisierung noch weitere Komprimierungstechniken. So kommen unter anderem beim H.264-Standard noch „Motion-Prediction“ und „Intra-Frame-Prediction“ zum Einsatz, die bei JPEG keine Verwendung finden. Damit ist eine Voraussage, welche Werte vom H.264 verändert werden schwieriger, als beim JPEG-Encoder. Zudem werden, wie in Kapitel 6.2.1 beschrieben, eine größere Blockgröße verwendet um eine höhere Streuungsmöglichkeiten zu erhalten. Im Gegensatz dazu konnten Eugen T. Lin et al. die klar definieren Blockgrößen des JPEG verwenden[18]. Das einfache Ignorieren von bestimmten Koeffizienten, wie es Eugen T. Lin. et al. bei JPEG vorschlagen, ist daher nicht möglich.

Stattdessen soll im hier vorgestellten Konzept untersucht werden, wie sich die Koeffizienten eines Blockes bei verlustbehafteter Kompression durch den x264 statistisch im Mittel verhalten. Aufbauend auf dieser Untersuchung soll anschließend wie in Kapitel 6.2.1 beschrieben, eine an diese Untersuchungen angepasste gewichtete Veränderung verwendet werden. Angelehnt an die Technik von Eugen T. Lin et al. sollen dabei jene Koeffizienten höher gewichtet werden, die im Mittel eine geringere Veränderung erfahren. Dazu wird zuerst eine grundlegende Gleichung ermittelt, die die Koeffizienten proportional zu ihrer statisch mittleren Größe gewichtet. Damit soll sichergestellt werden, dass kleine Koeffizienten nicht überproportional stark verändert werden. Eine solche überproportional starke Veränderung könnte sich auf den visuellen Einfluss niederschlagen. Anschließend soll die Gleichung so erweitert werden, dass sie die Koeffizienten stärker gewichtet die durch die verlustbehaftete Kompression im Mittel weniger stark verändert werden.

Verteilung der DCT-Koeffizienten

Um eine sinnvolle Gewichtung zu finden, wird als erstes untersucht, welche Größenordnung die einzelnen DCT-Koeffizienten im Durchschnitt besitzen. Dadurch lässt sich eine Gewichtung der DCT-Koeffizienten entwickeln, sodass die einzelnen DCT-Koeffizienten proportional zu ihrer durchschnittlichen Größe verändert werden.

Dass eine nicht an die Proportionen der Koeffizienten angepassten Gewichtung einen starken Einfluss haben kann, soll anhand eines Beispiels illustriert werden:

Gegeben sei ein 2×2 Pixel großer Block. Der durch die DCT transformierte Block sieht dabei wie in Abbildung 8.4 aus.

$$X = \begin{bmatrix} 100 & 30 \\ 30 & 0.5 \end{bmatrix}$$

Abbildung 8.4: Eine Beispielmatrix für die DCT-Koeffizienten

Bei einer gleichmäßig auf alle Koeffizienten verteilten Veränderung um den Wert vier, würden einige DCT-Koeffizienten verhältnismäßig stark und andere verhältnismäßig schwach verändert werden. Eine gleichmäßig verteilte Veränderung um den Wert vier bedeutet, bei vier Koeffizienten, eine Veränderung jedes der vier Koeffizienten um eins. Beim oberen linken Koeffizienten (x_{00}), würde dies einer Veränderung um ein Prozent entsprechen. Dagegen würde eine Veränderung des unteren rechten Koeffizienten (x_{11}) um denselben Wert einer Veränderung von 200 Prozent entsprechen. Eine solche starke Veränderung in den hohen Frequenzen wird visuell im Bild sichtbar werden.

Gesucht wird eine Gewichtung bei der die Veränderung der Koeffizienten proportional zu ihrem Einfluss auf das Bild ist. Um eine solche sinnvolle Gewichtung zu finden, werden die DCT transformierten Blöcke, der in Abschnitt 8 definierten Videos, addiert und die durchschnittlichen Werte summiert, die so entstehende Summe wird anschließend analysiert.

Da DCT-Koeffizienten sowohl positive als auch negative Werte annehmen können, wird hier der Mittelwert über die absoluten Werte der Koeffizienten gebildet. Das Ergebnis ist in Abbildung 8.5 dargestellt.

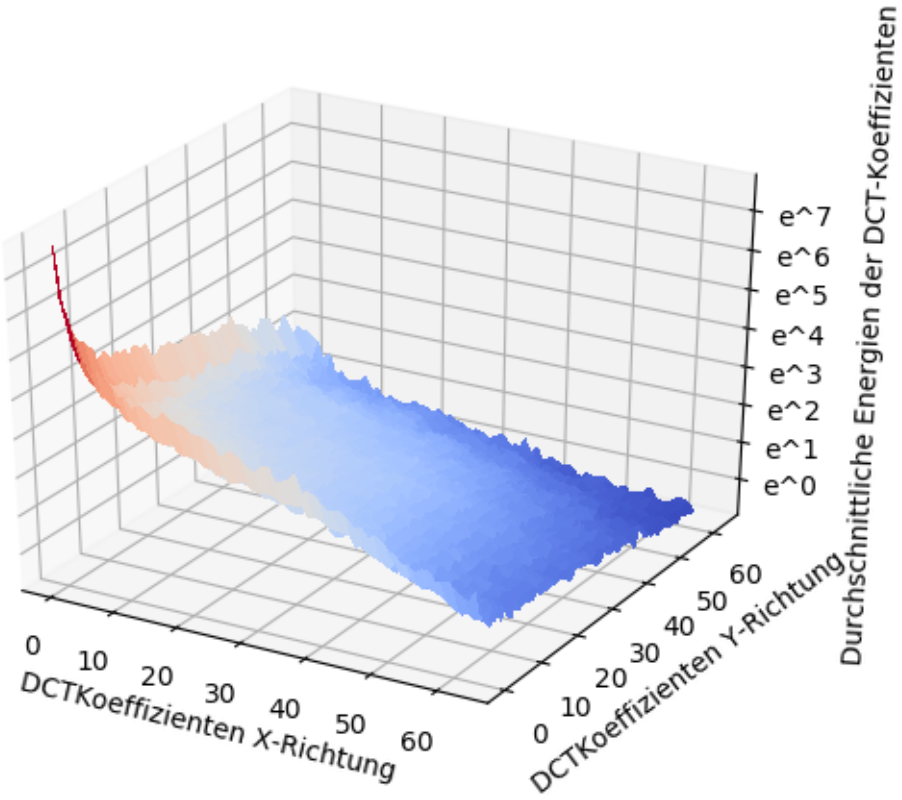


Abbildung 8.5: Der Mittelwert der absoluten Werte der DCT-Koeffizienten. Gemittelt sind alle Bilder der verwendeten Videosequenzen.

Die Darstellung zeichnet ein ähnliches Bild wie in Beispiel 8.4. Es zeigt sich, dass die Energien der Koeffizienten sehr ungleich verteilt sind, wobei die logarithmische Darstellung beachtet werden sollte.

Eine sinnvolle Gewichtung, die die Werte annähernd proportional gewichtet, könnte wie folgt aussehen:

$$e^{-(x+y)} \tag{8.2}$$

Würde man die oben genannte Flächengleichung in Abbildung 8.5 legen, würde auffallen, dass die Fläche die Koeffizienten nicht perfekt abbilden kann. Die niedrigen Frequenzen sind dabei weitaus stärker ausgebildet, als es die Gleichung 8.2 einfängt. Dies wurde im Rahmen dieser Arbeit aufgrund der zeitlichen Begrenzung nicht weiter untersucht und könnte Teil zukünftiger Untersuchungen sein.

Veränderung der DCT-Koeffizienten durch den Encoder

Neben dem visuellen Gesichtspunkt sollte jedoch auch beachtet werden, dass die hier vorgestellte Technik auch robust gegenüber verlustbehafteten Kompressionsverfahren sein soll.

Eine solche Robustheit ließe sich auch mit einer größeren Bucket-Größe erreichen, allerdings wird sich aus späteren Abschnitten noch ergeben, dass der Einfluss des Schwellwertes auf die Robustheit eher gering ausfällt. Daher muss die Robustheit primär durch die Gewichtung erreicht werden.

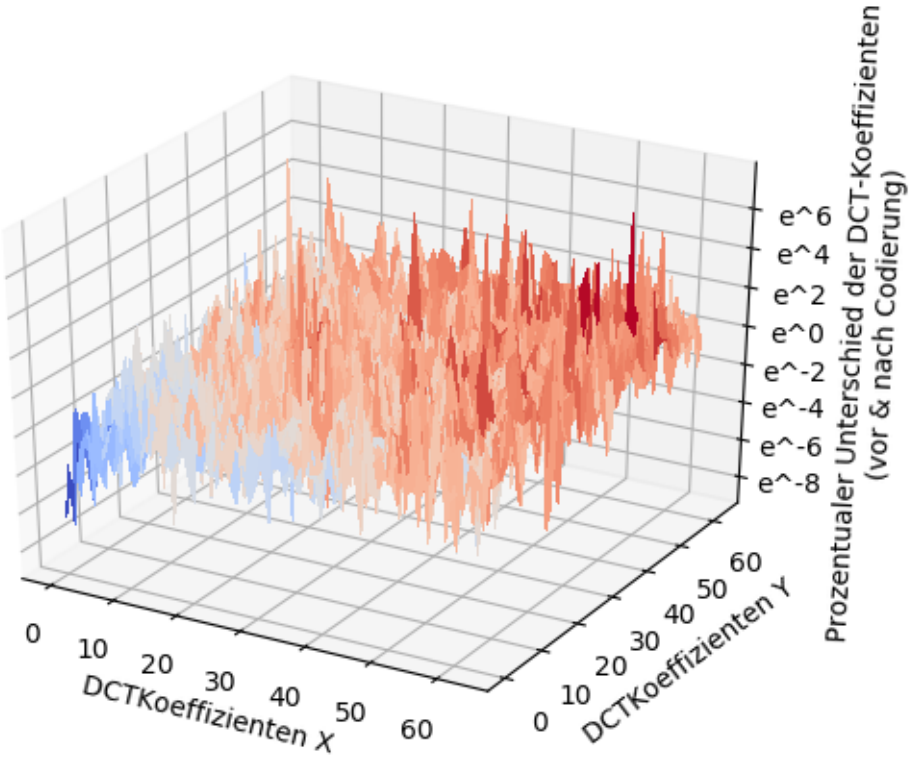


Abbildung 8.6: Die prozentuelle Abweichung der DCT-Koeffizienten vor und nach der Encodierung

Um eine robuste Gewichtung zu erreichen, wird im Folgenden analysiert, welche Arten von Veränderungen durch den Encoder am Bild vorgenommen werden. Abbildung 8.6 zeigt beispielhaft eine solche Veränderung, die durch den x264-Encoder mit einem QP von 40 entsteht. Wie in Kapitel 4 beschrieben wird sich im Rahmen dieser Arbeit auf QP Werte zwischen 20 und 40 eingegrenzt. Der Wert 40 wurde hierbei gewählt, da er den Extremfall abdeckt. Für die Darstellung wurde zuerst die prozentuale Abweichung der einzelnen Koeffizienten errechnet und anschließend der Durchschnitt über alle Blöcke gebildet. Die Ergebnisse sind auf einer exponentiellen Skala aufgetragen.

Abbildung 8.6 zeigt dabei, dass die niedrigen Frequenzen prozentual wenig und die hohen Frequenzen stärker verändert werden.

Durch die in Abschnitt 8.1.2 vorgestellte Gleichung 8.2 werden die Veränderungen auf alle Koeffizienten proportional zu ihrem statistisch mittleren Betrag gestreut. Dies berücksichtig die Veränderungen durch verlustbehaftete Kompression jedoch nicht, weshalb die so entstandene Gewichtung noch keine hohe Robustheit verspricht. Nachfolgend wird die Gleichung 8.2 weiterentwickelt, um neben der proportionalen Veränderung auch eine robuste Veränderung zu erlauben.

Es wird daher eine Gewichtung vorgeschlagen, welche niedrige Frequenzen deutlich stärker

gewichtet, als hohe Frequenzen. Eine geeignete quadratische Funktion in Form eines Paraboloids ist in Gleichung 8.3 gegeben und in Abbildung 8.7 visualisiert.

$$e^{-(x^2+y^2)\cdot\frac{1}{2\cdot a^2}}. \tag{8.3}$$

Es wird die in Gleichung 8.3 definierte Paraboloid verwendet, welche eine Funktion ähnlich, wie der in Abbildung 8.7 illustriert ist, ergibt.

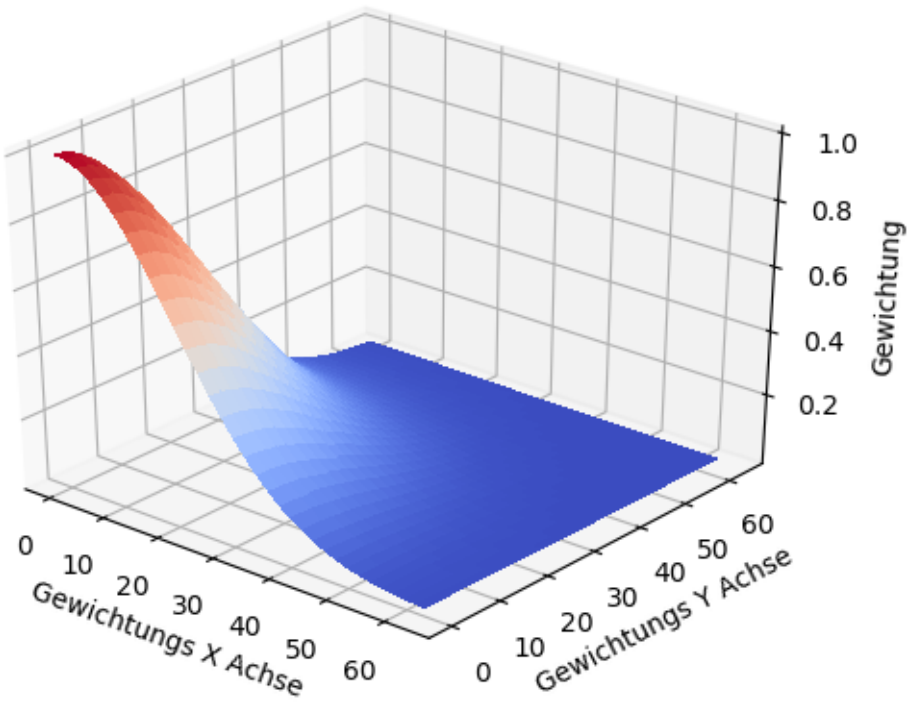


Abbildung 8.7: Eine vorgeschlagene Gewichtung nach der Formel $e^{-(x^2+y^2)\cdot\frac{1}{2\cdot 20^2}}$.

Die Steilheit der Gewichtungsfunktion kann über den Parameter a variiert werden. In Abschnitt 8.2 wird *a* verwendet um die Gewichtung so zu konfigurieren, dass sie einen optimalen Kompromiss zwischen Robustheit und Fehlererkennung bietet.

Dass die so gewählte Gewichtung eine signifikante höhere Robustheit zulässt ist in Abbildung 8.8 dargestellt. Dabei wurde ein Wasserzeichen einmal mit und einmal ohne Gewichtung eingebettet. Die Anzahl fehlerhaft detektierter Blöcke wurden auf der Y-Achse aufgetragen. Der Schwellwert wurde hierbei auf einen festen Wert ($T = 800$) gesetzt. Für *a* wurde im Fall der gewichteten Version der Wert 5 gewählt.

Dabei ist eindeutig zu sehen, dass durch die gewichtete Veränderung signifikant höhere Robustheit erreicht werden kann also durch eine nicht gewichtete Veränderung. Eine detaillierte Analyse der Robustheit ist in Kapitel 9.1.3 zu finden.

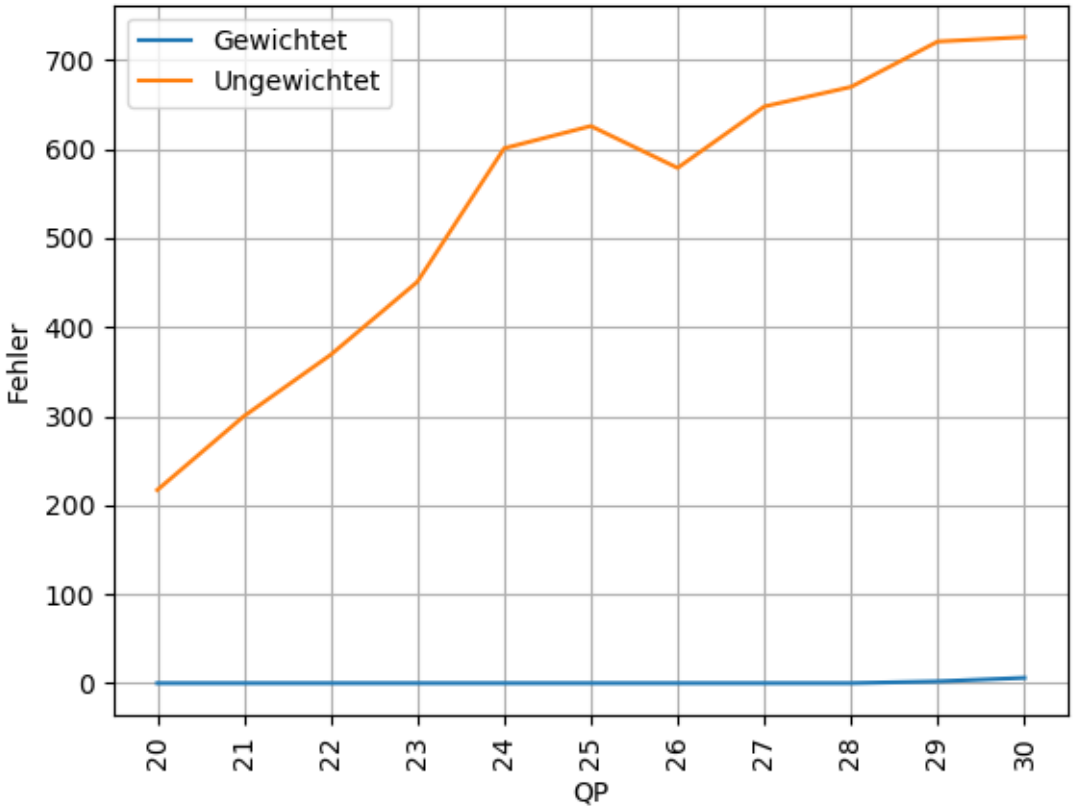
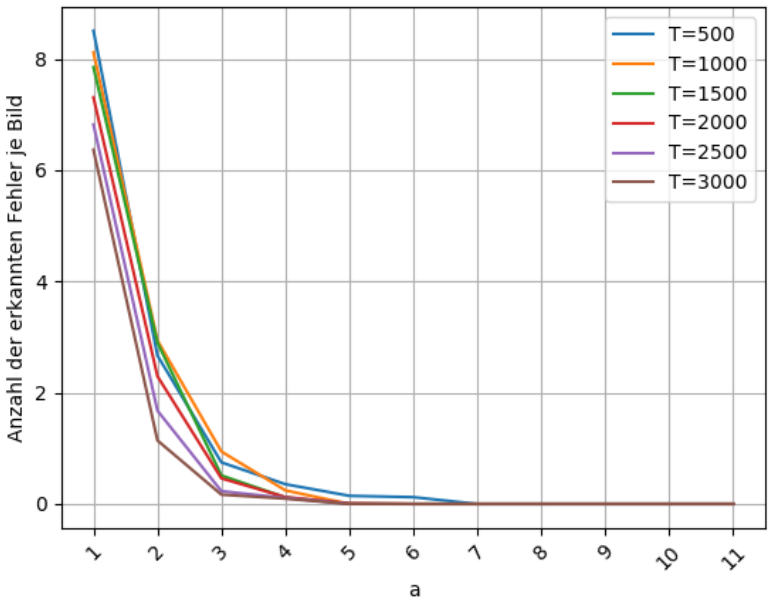


Abbildung 8.8: Vergleich von Robustheit mit und ohne Gewichtung ($T = 800$). Verwendet wurde das Video Akyio[26]

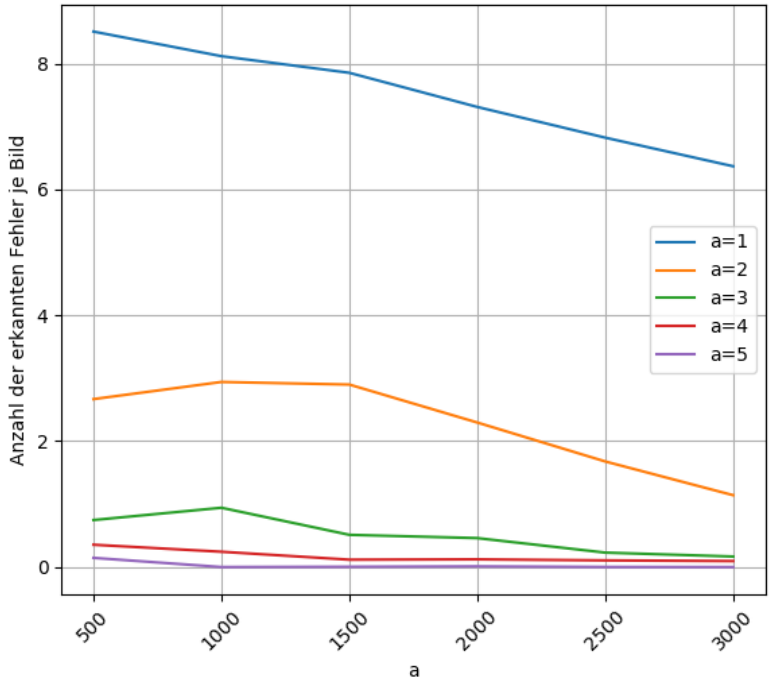
8.1.3 Der Schwellwert T

Ein weiterer Parameter, der Einfluss auf die Robustheit hat, ist der Schwellwert T. Das Encodieren eines Bildes mit dem x264-Encoder kann die Summe der DCT-Koeffizienten, wie bereits im Abschnitt 8.1.1 ermittelt wurde, verändern. Diese Veränderungen sollen durch die in Kapitel 6.2.2 beschriebenen Puffer abgefangen werden. Der Puffer ist dabei direkt abhängig von der Größe der Buckets und damit vom Schwellwert T. Sollte eine Veränderung stärker sein als der Puffer, wird die Summe in einen benachbarten Bucket verschoben und damit wird auch das encodierte Bit „gekippt". T muss demnach groß genug gewählt werden, um die Veränderungen der Koeffizienten durch den Encoder kompensieren zu können, aber klein genug, um durch die Einbettung des Wasserzeichens den Bildinhalt nicht zu stark zu verändern.

Um den Einfluss der Parameter T und a auf die Robustheit zu analysieren, wurden ein Wasserzeichen mit unterschiedlichen Werten für a und T eingebettet und durch den x264 codiert und decodiert. Anschließend wurde untersucht, aus wie vielen Blöcken der ursprüngliche Werte extrahiert werden konnte. Die Ergebnisse sind in Abbildung 8.9a dargestellt. Dabei wurde die Anzahl der fehlerhaft extrahierten Bits in Abhängigkeit des Parameters a dargestellt. Unterschiedliche Gewichtung wurden durch verschiedene Kurven repräsentiert. Je mehr Fehler erkannt werden, desto weniger Robust ist das Wasserzeichen bei den gewählten Parametern. Dabei ist zu erkennen, dass die Gewichtung einen weitaus höheren Einfluss auf die Robustheit hat, als die durch den Wert T kontrollierte Größe der Buckets.



(a) Als fehlerhaft erkannte Blöcke in Abhängigkeit der Gewichtung, bei einen QP von 35



(b) Als fehlerhaft erkannte Blöcke in Abhängigkeit des Schwellwertes, bei einen QP von 35

Abbildung 8.9: Zwei unterschiedliche Darstellungen für dieselben Daten: Die als fehlerhaft erkannten Blöcke nach der Encodierung mit dem x264 in Abhängigkeit von der Gewichtung (a) und in Abhängigkeit vom Schwellwert (b)

Um diesen Einfluss zu verdeutlichen, wurden dieselben Daten in Abbildung 8.9b in anderer Form dargestellt. Dabei wurde auf der X-Achse der Schwellwert T aufgetragen und die unterschiedlichen Gewichtungen wurden als einzelne Kurven dargestellt. Es zeigt sich, dass

unterschiedliche Werte für *a*, die Robustheit stark beeinflusst. So zeigt sich das für den Wert $a = 1$ noch relativ viele Fehler entstehen. Bei a=5 wurden bereits ab einem Schwellwert von $T = 1000$ keine Fehler mehr erkannt. Es zeigt sich auch, dass mit höherem Schwellwert auch die Robustheit erhöht werden kann, diese Zunahme jedoch relativ gesehen klein ist.

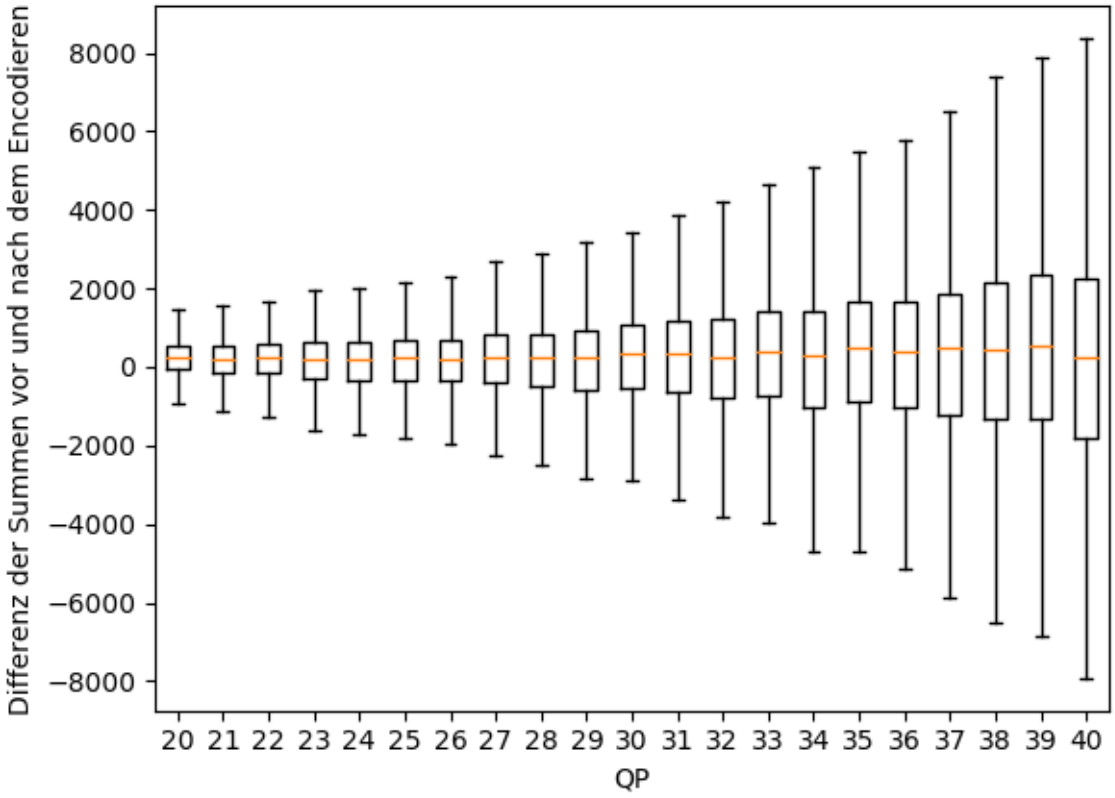


Abbildung 8.10: Die Differenz zwischen den unveränderten Bilddaten und den durch verlustbehaftete Kompression verarbeiteten Bilddaten. Hierbei wurde, um eine bessere Übersichtlichkeit zu erreichen, auf die Darstellung von Ausreißern verzichtet. Eine Version, die die Ausreißer mit einschließt, findet sich im Anhang in Abbildung A.1

Bei der verlustbehafteten Kompression werden viele Koeffizienten durch die Quantisierung leicht verändert. Diese kleinen Veränderungen akkumulieren sich, was sich in einer stark veränderten Gesamtsumme der Koeffizienten niederschlägt. In Abbildung 8.10 wurden die Veränderungen der Koeffizienten durch die verlustbehaftete Kompression dargestellt. Es fällt auf, dass die durch den Encoder entstehenden Veränderung an den Summen, stark variieren. Der Schwellwert müsste so groß sein, dass selbst die stärksten Veränderungen immer noch innerhalb des Buckets liegen. Deswegen kann durch das Variieren der Summe auch nur eine geringere Robustheit erreicht werden. Der Parameter *a* hingegen beeinflusst durch die Gewichtung den Einfluss der Veränderung durch den Encoder direkt. Daher hat *a* einen höheren Einfluss auf die Robustheit.

In Abbildung 8.9b zeigt sich außerdem, dass bei sehr geringen Werten für T teilweise weniger Fehler erkannt werden, als bei höheren Werten. Wie in Kapitel 6.2.2 definiert, sind die Buckets so angeordnet, dass zwei nebeneinanderliegende Buckets zwei unterschiedlichen binären Werten entsprechen. Durch Veränderungen am Bild wird die Summe so verschoben, dass sie im benachbarten Bucket landet. Findet eine ungewollte Verschiebung um mehr als $\frac{3}{2}T$ statt, kann es zum Überspringen eines Bucket kommen. Durch die in Kapitel 6.2.2 definierte alternierende

Anordnung der Buckets wird damit der selbe Wert extrahiert, der auch im ursprünglichen Bucket extrahiert worden wäre.

8.2 FEHLERERKENNUNG

In diesem Abschnitt soll untersucht werden, welchen Einfluss Gewichtung und Schwellwert auf die Fehlererkennung haben. Dabei wird hier auf die Erkennung von Bildfehlern eingegangen. Wie in Kapitel 6.3 beschrieben, werden Fehler anhand der extrahierten Daten erkannt. Das Erkennen dieser Fehler ist also nicht direkt von der Gewichtung und dem Schwellwert abhängig, sondern von der Fähigkeit, die Daten fehlerfrei extrahieren zu können. Das Ergebnis der zeitlichen Fehlererkennung wird im Kapitel 9 behandelt.

Die Fehlererkennung soll aufbauend auf der Robustheit untersucht werden, d.h. die in Abschnitt 8.1.3 gewählte Gewichtung findet weiterhin Anwendung. Im folgenden Abschnitt soll der Einfluss der Parameter T und a auf die Fehlererkennungseigenschaften des Wasserzeichens untersucht werden. Anschließend werden die Werte für T und a so gewählt, dass sie weiterhin robust gegenüber Veränderungen durch den Encoder sind, gleichzeitig jedoch auch eine hohe Fehlererkennung besitzen.

Dafür wurde das Einbettungsalgorithmus mit unterschiedlichen Parametern versehen und Blockfehler in das Bild eingefügt. Anschließend wurde untersucht wie viele Fehler vom Prüfsystem erkannt wurden. Dafür wurden gleichmäßig über jedes Bild 20 Blockfehler eingefügt, und ein QP von 20 verwendet. Die Ergebnisse für unterschiedliche Parameter a und T sind in Abbildung 8.11 dargestellt.

Die erkannten Fehler nehmen für alle a mit steigendem Schwellwert T stark zu, sinkt jedoch danach wieder stark ab. Dass die Fehlererkennung erst einmal stark zunimmt scheint kontraintuitiv. Es hängt jedoch mit dem bereits in Abschnitt 8.1.3 erwähnten „überspringen“ von Buckets zusammen. Dabei wird die Summe durch den Encoder verändert, die Buckets sind allerdings so klein, dass die Verschiebung den benachbarte Bucket überspringt. Der übernächste Bucket codiert, wie in Kapitel 6.2.2 beschrieben wieder den ursprüngliche Wert somit kann beim extrahieren kein Fehler erkannt werden.
Ab einem gewissen Wert nimmt die Fehlererkennung wieder ab, da die Buckets so groß werden, dass die durch die Fehler hervorgerufene Veränderung nicht mehr ausreichend ist um die Summe aus dem Ausgangs-Bucket zu schieben.

Generell lässt sich mit allen getesteten Werten für a eine ähnlich hohe Fehlererkennungseigenschaft erreichen, die Fehlerkennungseigenschaften zu einem gegebenen a hängt dabei stark vom verwendeten Schwellwert ab. Aus Abbildung 8.11 lässt sich zudem entnehmen, dass mit nahezu allen Werten für a eine gute Fehlererkennung erreicht werden kann, sofern der zum a passend gewählte Wert für T gewählt wurde. Dabei zeigt sich dass die Fehlererkennung besonders im Bereich zwischen $T = 600$ und $T = 800$ maximiert. Dieser Bereich ist in Abbildung 8.11 noch einmal detailliert dargestellt. Abbildung 8.12 zeigt, dass $a = 5$ eine hohe Fehlererkennung erreichen kann und im dargestellten Bereich relativ stabil hohe Fehlererkennungseigenschaften besitzt. Aus der in Abschnitt 8.1.3 gezeigten Abbildung 8.9, erscheint eine hoher Robustheit für $a = 5$ erreicht zu werden, wenn $T = 800$ verwendet wird. Für die folgenden Untersuchungen werden daher die Werte a = 5 und T = 800 verwendet. Zwar bietet der Parameter $a = 6$ eine minimal höhere Fehlererkennung und höherer Robustheit, allerdings bedeutet ein höherer Wert für a auch immer eine auf niedrige Frequenzen fokussierte Gewichtung. Die Hypothese ist, dass durch eine weniger stark fokussierte Gewichtung die Fragilität gegenüber anderen als den hier vorgestellten Fehlern noch einmal zunimmt. Daher wird für a hier der Wert 5 verwendet.

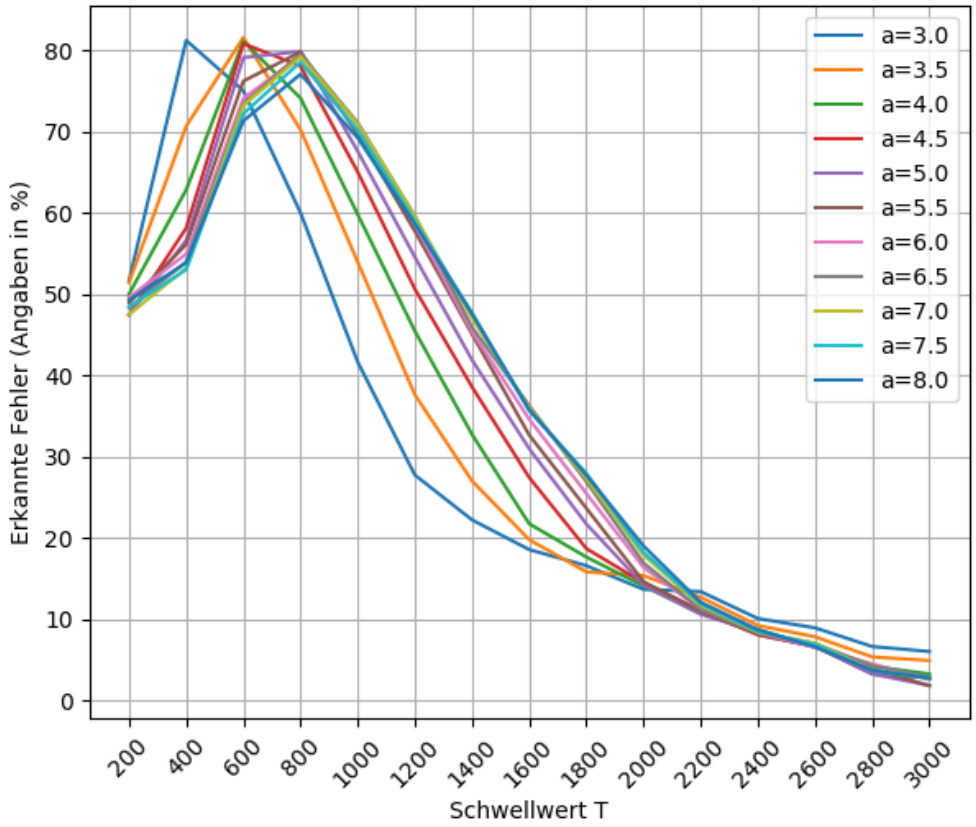


Abbildung 8.11: Die Blockgröße und ihre Fehlererkennungseigenschaften, bei unterschiedlichen Werten für *a* und *T*

8.3 VISUELLER EINFLUSS

Im folgenden Abschnitt wird noch untersucht, welchen visuellen Einfluss die vorher definierten Parameter auf das Bild haben. Die Gewichtung, wie sie im Abschnitt 8.1 erarbeitet wurde, erhöht (bzw. senkt) alle DCT-Koeffizienten. Dieses Anheben bzw. Absenken kann zu Problemen führen, da die Struktur der DCT-Funktion nicht zufällig ist, sondern bestimmten Mustern folgt. Dafür ist die verwendete DCT-Funktion in Abbildung 8.4 noch einmal dargestellt. [3]

$$y(k) = 2 \cdot \sum_{n=0}^{N-1} x[n] \cdot cos(\pi \cdot k \cdot \frac{2n+1}{2 \cdot N}), 0 <= k < N. \tag{8.4}$$

Dabei ist *N* eine Dimension der Blockgröße, *k* die Position des aktuellen Pixelwertes und *x* ist der Pixelwert selbst.

Da *k* im Falle des ersten Pixels immer zu null evaluiert, wird die Kosinusfunktion immer zu 1 evaluiert. Damit ist die Amplitude für alle Koeffizienten an dieser Stelle stets ungleich 0. Jeder Pixel eines Blocks wird bei der Rücktransformation aus der Summe aller Kosinusfunktionen rekonstruiert. Somit entsteht bei der Erhöhung aller DCT-Koeffizienten in der linkeren oberen

[3] Zur besseren Illustration ist hier lediglich die eindimensionale Funktion dargestellt. Bei einer zweidimensionalen DCT, wie sie bei der Verarbeitung von Bildern zum Einsatz kommt, werden die Daten zweimal DCT-transformiert, einmal in horizontaler und einmal in vertikaler Richtung.

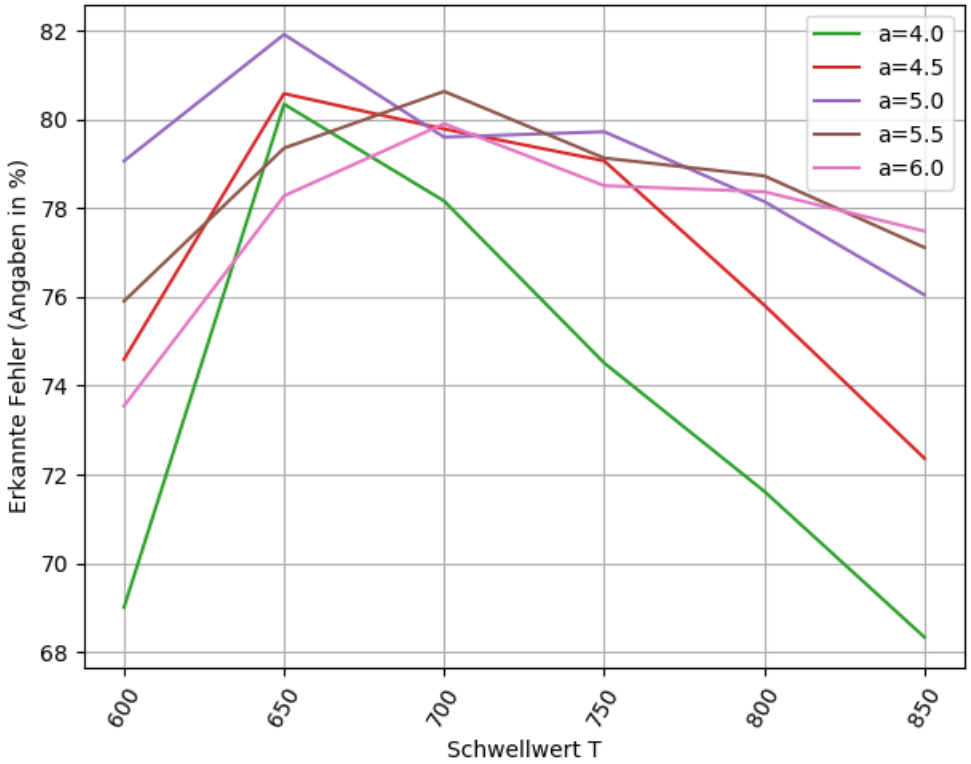


Abbildung 8.12: Eine ausschnitthafte Darstellung der Blockgröße und ihre Fehlererkennungseigenschaften, bei unterschiedlichen Werten für *a* und *T*

Ecke häufig Artefakte. In Abbildung 8.13 wurde eine aus dieser Formel entstehende DCT-Frequenzen noch einmal visualisiert, und die problematischen Stellen wurden rot eingerahmt.

Wenn man alle Koeffizienten kollektiv erhöht, auch wenn dies gewichtet geschieht, erhält man in der oberen linken Ecke des Blockes Artefakte. Je nach Vorzeichen des entsprechenden Koeffizienten entweder einen dunklen oder einen hellen Bereich. Diese Artefakte sind in Abbildung 8.14a ausschnitthaft auf dem Video „Foreman“ dargestellt.

Um dies zu verhindern wird vorgeschlagen, die Gewichtung zu erweitern. Dabei wird durch normal verteilte Werte mit der vorher definierten Gewichtung multipliziert. [4] Die so entstehende Gewichtung erhöht die Frequenzen nicht alle in gleicher Weise. Stattdessen werden einige Werte positiv verändern und andere negativ.

Wie Abbildung 8.14 zeigt lassen sich durch das Multiplizieren der vorhergehenden Gewichtung mit einer Gewichtungsmatrix mit normal verteilten Werten die sichtbaren Artefakte erheblich verringern. Es ist darauf hinzuweisen, dass diese Optimierung bereits im gesamten Analysekapitel in den Untersuchungen angewendet wurde. Eine Subjektive Untersuchung des Einflusses wird in Kapitel 9.3.2 durchgeführt.

[4] Die Werte werden mithilfe des Python Befehls: np.random.normal(size=(64,64)) durchgeführt

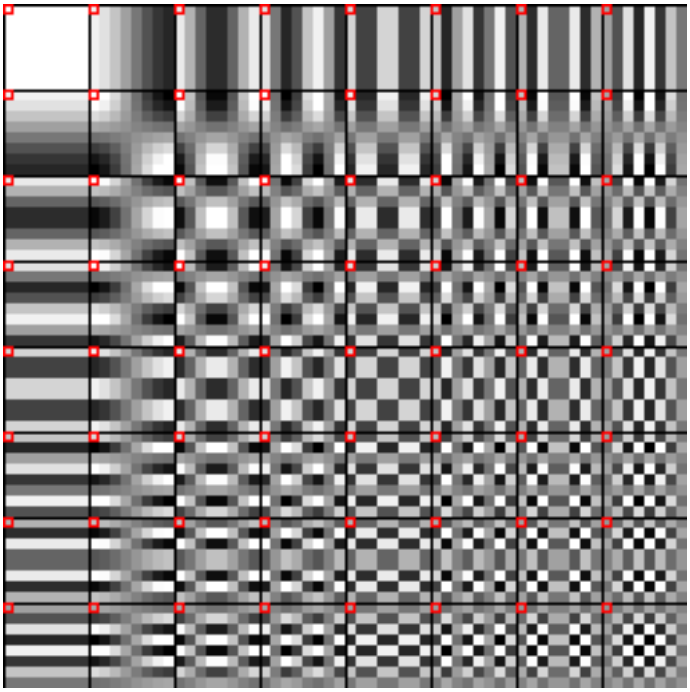


Abbildung 8.13: Die Visualisierung der DCT-Frequenzen

8.4 ZUSAMMENFASSUNG

In diesem Kapitel wurde der Einfluss diverser Parameter auf das in Kapitel 6 entwickelte Konzept untersucht. Dabei waren die Metaziele - Robustheit, Fehlererkennung und visueller Einfluss - Mittelpunkt der Untersuchungen.

Angefangen wurde hierbei bei der Definition der optimalen Blockgröße. Hierbei wurde eine Blockgröße von 64×64 identifiziert da sie durch die Möglichkeit zur Einbettung von Redundanzinformationen eine hohe Robustheit bietet. Anschließend wurde eine Gewichtung gesucht, die eine hohe Robustheit und einen geringen visuellen Einfluss hat. Die vorgeschlagene Gleichung beschreibt die gewählte Gewichtung:

$$e^{-(x^2+y^2)\cdot\frac{1}{2\cdot a^2}}.$$

Dabei werden vor allem die niedrigen Frequenzen stark gewichtet, weil sich im Rahmen der Analyse ergeben hat, dass diese durch den Encoder verhältnismäßig schwach verändert werden. Über den Parameter *a* lässt sich die Steilheit der Fläche variieren, was dazu verwendet, wurde um die Gewichtung auf Fehlererkennung zu optimieren.

Anschließend wurde aufbauend auf die vorhergehende Gewichtung untersucht, welchen Einfluss das Variieren des Parameter *a* und der Bucket-Größe über den Parameter T auf die Robustheit des Wasserzeichens hat. Dabei wurde herausgefunden, dass besonders der Parameter *a* einen starken Einfluss auf die Robustheit hat. Dabei konnte durch eine Wahl von $a = 5$ eine hohe Robustheit gegenüber Veränderungen der verlustbehafteten Kompression erreicht werden.

Anschließend wurde die Fehlererkennungseigenschaften des Konzeptes untersucht, hierbei wurden ebenfalls die Parameter *a* und der Schwellwert *T*, der die Größe der Buckets definiert, untersucht. Eine hohe Fehlererkennungseigenschaft konnte für alle untersuchten Werte für *a*

(a) Ein Beispiel für die Einbettung eines Wasserzeichens, ohne die zusätzliche Gewichtung

(b) Dasselbe Bild mit der zusätzlichen Gewichtung.

Abbildung 8.14: Der Unterschied des Wasserzeichens mit und ohne der in Abschnitt 8.3 definierten zusätzlichen Gewichtung.

erreicht werden, in dem der Schwellwert T abhängig von a in einem Bereich zwischen 400 und 800 gewählt wurde. Um einen guten Kompromiss zwischen Fehlererkennung und Robustheit zu erreichen, wurden die Werte $a = 5$ und $T = 800$ gewählt. Abschließend wurde die Gewichtung noch um einen zufälligen Anteil erweitert, um den Einfluss der Artefakte signifikant zu vermindern.

9 AUSWERTUNG DER ERGEBNISSE

AUSWERTUNG DER ERGEBNISSE

Im folgenden Kapitel wird ermittelt, welche der in Kapitel 5 definierten Ziele mit dem in Kapitel 6 entwickelten und im Kapitel 8 optimierten Konzept erreicht werden können. Dabei sollen auch dessen Limitationen beleuchtet werden. Für die Auswertung in diesem Kapitel wurde eine andere Auswahl an Videos getroffen, als in den vorherigen Kapiteln. So soll gezeigt werden, dass die erarbeiteten Methoden unabhängig von den gewählten Videos funktionieren.

Es fand in diesem Kapitel wieder eine Auswahl an Videos aus der Xiph-Datenbank[26] Verwendung. Wie schon im Kapitel 8 wurde darauf geachtet, Videos mit Videokonferenz-ähnlichen Situationen zu verwenden. Die verwendeten Videos sind in der Datenbank unter folgenden Namen zu finden: „Johnny", „vidyo1", „vidyo3" und „vidyo4". Alle diese Videos haben eine Auflösung von 1280×720 Pixeln. Diese Auflösung wurde gewählt, da es sich um die Minimalauflösung handelt, für die der Algorithmus konzeptioniert wurde, wie es Kapitel 8.1.1 zu entnehmen ist. Um trotzdem sicherzustellen, dass der Wasserzeichen-Algorithmus unabhängig von der verwendeten Auflösung funktionstüchtig ist, wird in Abschnitt 9.1.2 eine Analyse der Skalierbarkeit des Wasserzeichens in Abhängigkeit von unterschiedlichen Auflösungen durchgeführt.

Das weitere Kapitel ist folgendermaßen aufgebaut:

Als Erstes werden die allgemeinen Eigenschaften des Wasserzeichens vorgestellt. Hierbei wird sowohl auf den visuellen Einfluss, als auch auf Robustheit und Auslösungs-Skalierbarkeit eingegangen. Darauffolgend werden die Fehlererkennungseigenschaften des Wasserzeichens ermittelt. Abschließend wird noch der visuelle Einfluss des Wasserzeichens untersucht.

9.1 ALLGEMEINES

Im folgenden Abschnitt sollen die allgemeinen Eigenschaften des Wasserzeichens beleuchtet werden. Als Erstes werden die Einflüsse des Wasserzeichens auf das Video vorgestellt. Dabei wird erst auf das Einzelbild eingegangen und anschließend wird untersucht, welchen Einfluss das Wasserzeichen bei aufeinanderfolgenden Bildern hat. Als Nächstes wird gezeigt, dass die Einbettung der Bits unabhängig von der gewählten Auflösung ist. Danach wird die Robustheit des Wasserzeichens untersucht.

9.1.1 Auswirkung auf den Bildinhalt

In diesem Abschnitt werden beispielhaft einige durch das Wasserzeichen veränderte Bilder vorgestellt und ihr Einfluss beim Abspielen als Video dargestellt.

Abbildung 9.2 zeigt zwei im Video aufeinanderfolgende Bilder. Dabei sind vor allem die Artefakte auf dem dunklen Hintergrund auf der linken Seite, hinter der Person, auffällig. Dies sind Bereiche, die im Originalvideo wenig Struktur besitzen, wodurch es relativ wenige Frequenzen gibt, die vom Wasserzeichen verändert werden konnten.

Für die notwendige Verschiebung der DCT-Koeffizienten bei der Einbettung des Wasserzeichens mussten die wenigen vorhandenen bzw. schwach ausgeprägten Frequenz-Koeffizienten somit stärker verändert werden. Dies ist in Abbildung 9.2 in den unterschiedlich stark strukturierten Bereichen des Hintergrunds gut nachzuvollziehen. In den Regionen des Bildes, die mehr

Abbildung 9.1: Ein originales unverändertes Beispielbild aus den Videosequenzen.

(a) Bild n

(b) Bild n+1

Abbildung 9.2: Zwei aufeinanderfolgende Bilder aus den Videosequenzen, die mit dem Wasserzeichen versehen wurden.

Strukturen besitzen, ist das Wasserzeichen weniger sichtbar. Um dies zu verdeutlichen ist in Abbildung 9.4 ein Ausschnitt des Bildes dargestellt, wobei lediglich der Kopf und der stark strukturierte Hintergrund dargestellt wurden. Bei der Betrachtung dieses Ausschnittes sind die Übergänge zwischen den Artefakten besonders im strukturierten Hintergrund visuell weniger stark wahrnehmbar.

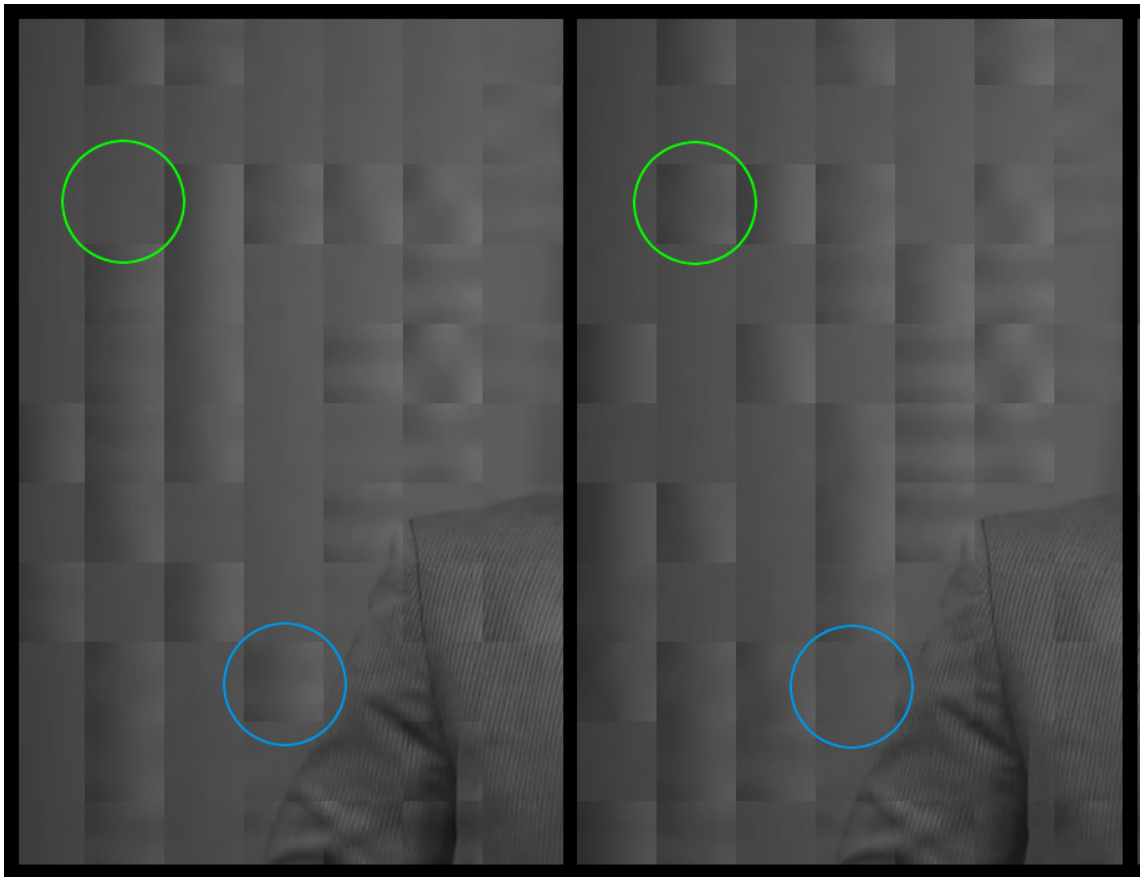


Abbildung 9.3: Eine Illustration der Auswirkung des Wasserzeichens auf ein Video. Dazu sind die im Video aufeinanderfolgenden Bilder aus Abbildung 9.2 ausschnitthaft dargestellt und zwei besonders auffällig veränderte Regionen von einem auf das nächste Bild sind farblich paarweise hervorgehoben.

Was bei der Einzelbild-Betrachtung jedoch nicht auffällt, ist, dass sich bei zwei aufeinanderfolgenden Bildern teilweise die Ausprägung der Artefakte von einem auf das nächste Bild verändert. Dies geschieht genau dann, wenn im Bild n in dem gegebenen Block ein anderer Wert V eingebettet wird, als im gleichen Block in Bild $n+1$, mit $n \in \mathbb{N}, V \in 0{,}1$. Im Video prägt sich dieses als eine Art „Flackern“ aus, also ein ständiger Wechsel des Bildinhalts an der besagten Stelle. Dieses Phänomen wird in Abbildung 9.3 dargestellt. Dabei wurden Ausschnitte der aus Abbildung 9.2 aufeinanderfolgenden Einzelbilder nebeneinander gelegt und zwei sich besonders stark ändernde Regionen jeweils paarweise in unterschiedlichen Farben hervorgehoben. Wie störend dieses Flackern auf den Betrachter wirkt, wird in Abschnitt 9.3.2 durch einen subjektiven Test untersucht.

9.1.2 Auflösungs-Skalierbarkeit

Dass der entwickelte Algorithmus zum Einbetten von Bits nicht nur auf eine bestimmte Auflösung beschränkt ist, soll im folgenden Abschnitt gezeigt werden. Dazu wurde ein Video mit ffmpeg in sechs unterschiedliche Auflösungen skaliert. Anschließend wurde das Wasserzeichen mit fester Blockgröße in die Videos eingebettet und untersucht, wie viele Bits sich erfolgreich bei den unterschiedlichen Auflösungen extrahieren ließen. Als Ausgangspunkt wurde ein Video mit der Auflösung 1920×1080 gewählt („pedestrian_area“ aus der Xiph-Datenbank[26]), wodurch sich folgende Zielauflösungen ohne Interpolation von Zwischen-Pixeln erzeugen lassen:
160×90, 480×272, 640×360, 960×540, 1280×720, 1920×1080.

Abbildung 9.4: Ein Ausschnitt aus dem Video „Johnny“ mit stärker strukturierten Regionen. Das Wasserzeichen ist insbesondere im Haarbereich und im stark strukturierten Hintergrund weniger sichtbar.

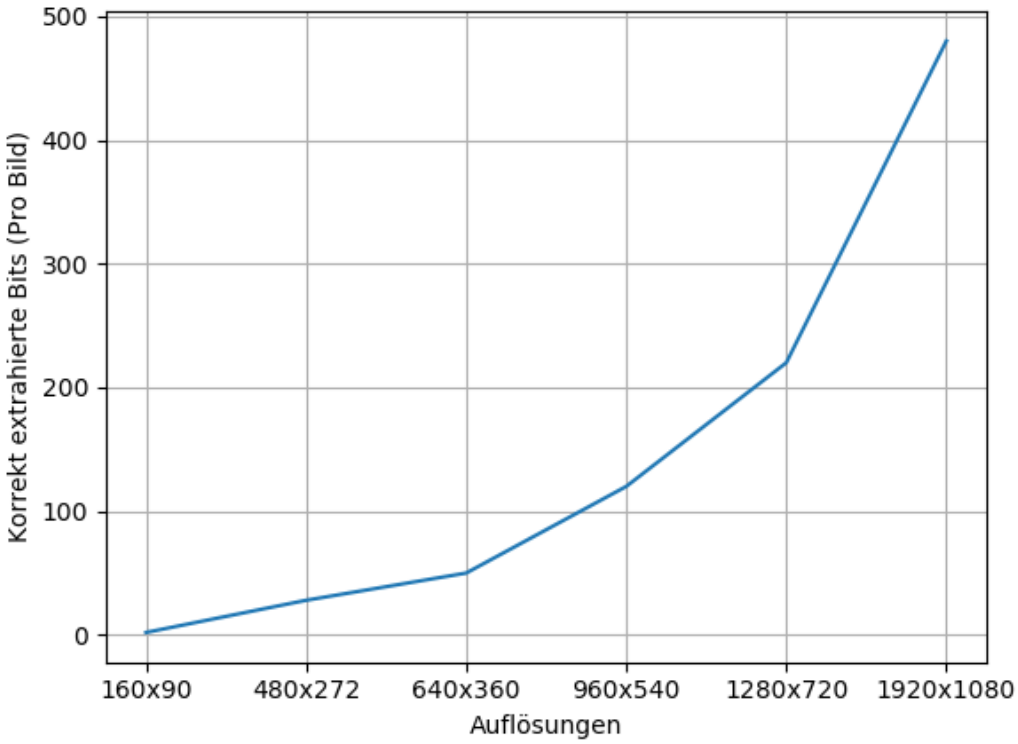


Abbildung 9.5: Die Einbettungskapazität des Wasserzeichens bei unterschiedlichen Auflösungen.

Das Ergebnis dieser Analyse ist in Abbildung 9.5 zu sehen, es wurde hierbei die Einbettungskapazität in Bits in Abhängigkeit von der Pixelanzahl dargestellt. Dabei konnten alle eingebetteten Bits auch wieder korrekt extrahiert werden, sodass die hier ermittelte Anzahl auch der maximal möglichen Anzahl entspricht. Durch die gleichbleibende Blockgröße erhöht sich mit steigender Auflösung die Anzahl der einbettbaren Bits. Anzumerken ist noch, dass die Einbettungskapazität des Algorithmus nicht kontinuierlich mit der Pixelanzahl zunimmt, sondern sich nur bei einer Zunahme der horizontalen bzw. vertikalen Pixelanzahl um die volle Blockgröße (hier 64) erhöht.

Generell zeigt sich, dass der Einbettungsalgorithmus größtenteils unabhängig von der Auflösung arbeiten kann. Um die notwendigen Fehlerkorrektureigenschaften, die im Kapitel 6.3 beschrieben sind, zu erhalten, wird eine minimale Anzahl an Bits benötigt. Dies ist bei kleineren Auflösungen jedoch nicht gegeben. Somit kann das Wasserzeichen zwar bei beliebigen Videos zum Einbetten von Informationen verwendet werden, um die Fehlererkennungseigenschaften jedoch so zu verwenden wie vorgesehen, ist eine minimale Auflösung notwendig. Die minimale Auflösung, aus der oben definierten Auswahl, die alle Eigenschaften besitzt, die in Kapitel 8.1.1 definiert wurden, ist 960×540.
Über die Variation der Blockgröße ließe sich die Einbettungskapazität auf Kosten der Robustheit erhöhen. Somit ließen sich dieselben Fehlerkorrektureigenschaften auch bei geringeren Auflösungen erreichen. Die gewählte Blockgröße von 64×64 Pixeln hat sich im Rahmen der Untersuchungen allerdings als guter Kompromiss zwischen Einbettungskapazität und Robustheit bewährt. Alternativ könnte man bei gleicher Blockgröße die Informationsbits des Wasserzeichens in mehrere Gruppen einteilen und sie in mehrere aufeinanderfolgende einzelne Bilder einbetten.

9.1.3 Robustheit

Wie bereits im Konzept herausgestellt, ist es essentiell, dass das System robust gegenüber der Encoder-Decoder-Kette ist. In diesem Abschnitt soll nun herausgestellt werden, inwiefern sich das Metaziel Robustheit mit dem gegebenen System erfüllen lässt.

Dafür wurden die Videos mit dem Wasserzeichen markiert und danach mit unterschiedlichen Quantisierungsparametern (QP) encodiert. Der QP-Parameter bestimmt im H.264-Standard die Schrittweite der Quantisierung und steuert, vereinfacht dargestellt, den Grad des Informationsverlusts bei der Kompression (hohe QP-Werte entsprechen einer starken Quantisierung). Er ist somit die wesentliche Stellgröße für die Stärke der durch den Encoder eingeführten Veränderungen. Im Rahmen dieser Arbeit wurden, wie in Kapitel 4 festgelegt, QP-Werte im Bereich 20 bis 40 verwendet. Eine genauere Beschreibung zum Encoder findet sich in Abschnitt 2.3. Die so mit dem Wasserzeichen versehenen und encodierten Videos wurden anschließend decodiert und die über alle Videos durchschnittlich fehlerhaft übertragenen Bits wurden in Abhängigkeit vom QP in Abbildung 9.6 dargestellt.

Für die Ermittlung der Robustheit wurden die in Kapitel 8 festgelegten Werte für die Parameter verwendet, das heißt: $a = 5$ und $T = 800$. Abbildung 9.6 zeigt den prozentualen Anteil fehlerhaft detektierter Blöcke bei unterschiedlichen QP-Werten gemittelt über alle Testvideos. Im Rahmen der Untersuchung stellte sich heraus, dass bei QPs über 28 einzelne Bilder so stark verändert wurden, dass die Fehlerkorrektureigenschaften des BCH-Codes von 30 Bits nicht mehr ausreichten, um die Fehler zu lokalisieren und zu korrigieren. Damit waren keine zweifelsfreien Aussagen mehr über die Anzahl der fehlerhaft extrahierten Bits möglich, daher wurden in Abbildung 9.6 lediglich Werte bis zu einem QP von 28 dargestellt.

Es ist zu erkennen, dass die Kurve exponentiell ansteigt. Zu diesem Zweck wurde mithilfe einer „exponentiell fit"-Funktion eine Kurvengleichung errechnet. Eine Annäherungskurve kann durch

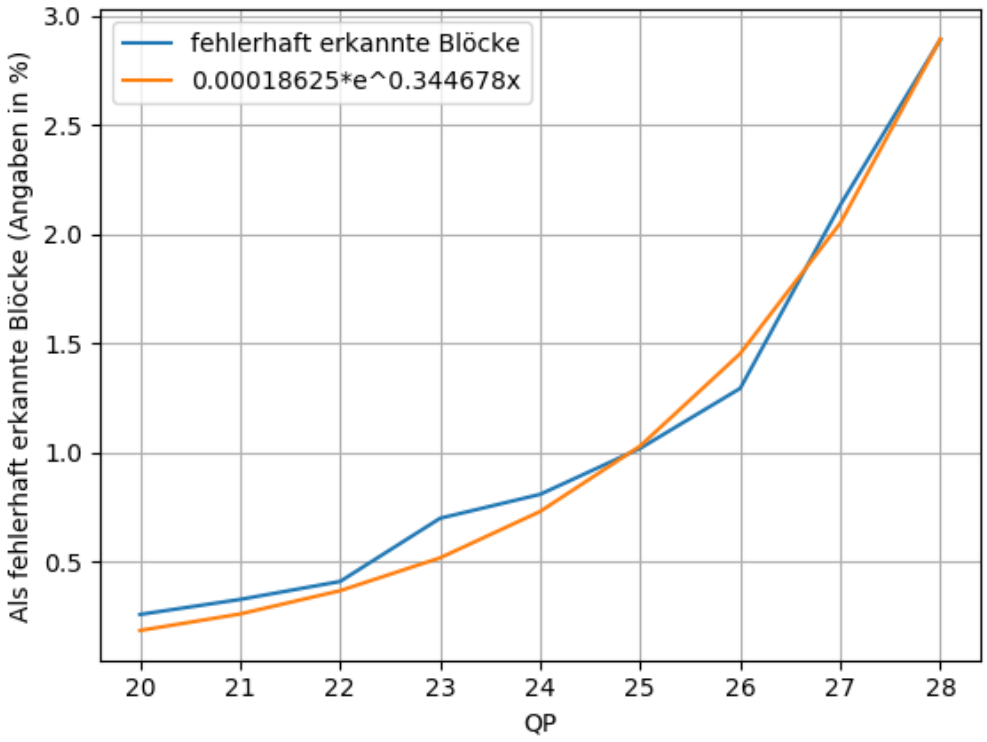


Abbildung 9.6: Die als Fehler erkannten Blöcke in Prozent in Abhängigkeit des QPs

die Gleichung 9.1 beschrieben werden.

$$0.00018625 * e^{0.344678x} \tag{9.1}$$

Es ist also festzuhalten, dass die vorgestellte Technik robust gegenüber schwachen und mittelstarken Kompressionsraten ist. Darüber hinaus lässt sich mit den gewählten Parametern keine Robustheit mehr erreichen.
Die Ergebnisse aus Kapitel 8 lassen die Vermutung zu, dass mit einem anderen Wert für *a* eine höhere Robustheit erreicht werden könnte. Die isolierte Betrachtung der Robustheit war jedoch nicht Fokus dieser Arbeit. Dies könnte in späteren Arbeiten noch weiter untersucht werden.

9.2 FEHLERERKENNUNG

Im folgenden Abschnitt soll das zweite Metaziel - Fehlererkennung - untersucht werden. Dabei werden die bereits in Kapitel 6 definierten Fehler untersucht. Angefangen wird mit den Bildinhaltsfehlern, anschließend werden die zeitlichen Fehler untersucht und abschließend die Bitstromfehler.

9.2.1 Bildinhaltsfehler

Ein zentrales Ziel bei der Fehlererkennung ist, dass Fehler, wie in Kapitel 6 definiert, im Bild erkannt werden können. Um die Fehlererkennung zu gewährleisten, wurde eine Videosequenz

gewählt und es wurden in jedem Bild Fehler eingebettet. Die Einbettung wurde durch Skripte realisiert, welche das Bild innerhalb der Prozesskette verändern. Die Details dieser Skripte finden sich im Kapitel 7. Im Rahmen dieser Arbeit wurden die Bildinhaltsfehler getrennt voneinander untersucht, eine Überlagerung mehrerer Fehlerklassen in einer Analyse findet nicht statt. Im Folgenden werden die Ergebnisse der Fehlererkennung vorgestellt.

Blockfehler

In diesem Abschnitt werden die Ergebnisse zur Blockfehlererkennung präsentiert. Hierzu wurde in jedes Bild der Videosequenzen ein Blockfehler eingefügt. Die Position jedes Blockfehlers wurde in jedem Bild variiert, beim Einfügen der Blockfehler wurde ihre Größe verändert. Um eine Reproduzierbarkeit zu erlangen und sicher zu sein, dass das System auch alle Teile des Bildes testet, wurden die Fehlerblöcke zufällig, aber reproduzierbar auf dem Bild positioniert. Die genaue Ermittlung der Positionierung ist im Kapitel 7.2.1 beschrieben.

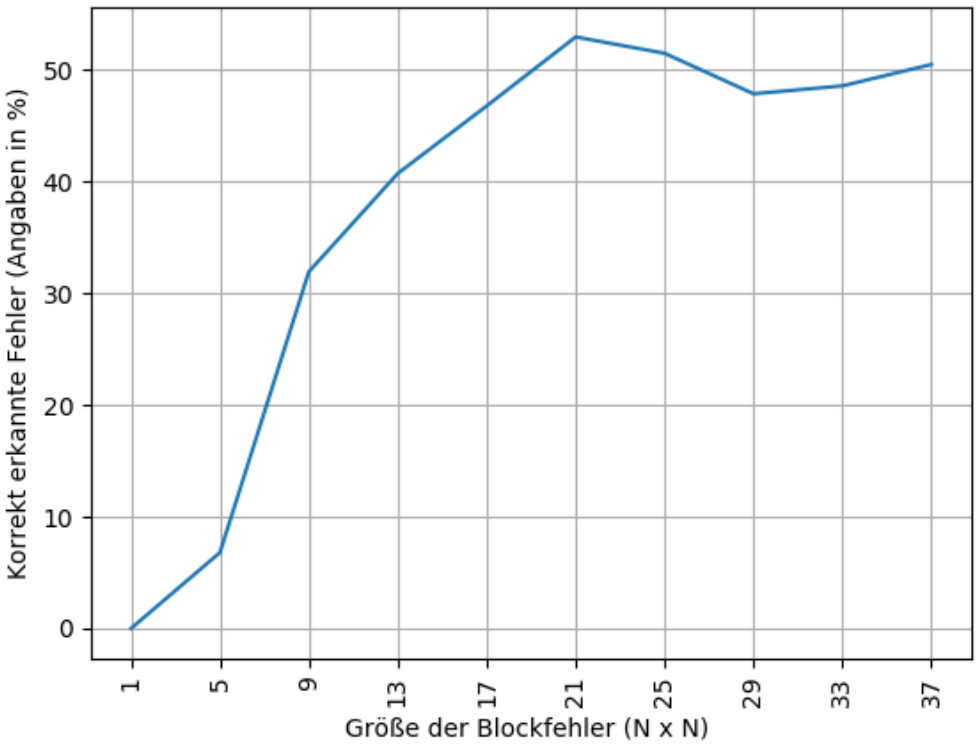


Abbildung 9.7: Die Anzahl der erkannten Blockfehler in Abhängigkeit der Größe der Blockfehler in Pixel.

Bei der Analyse wurde die erwartete Position mit der Blockposition des aufgetretenen Fehlers verglichen. Wie sich die Fehlererkennung bei unterschiedlichen Blockgrößen entwickelt, ist in Abbildung 9.7 dargestellt.

Aus Abbildung 9.7 geht hervor, dass das entwickelte System nicht in der Lage ist, sehr kleine Blockfehler zuverlässig zu erkennen, aber bei einer Blockgröße von 9 lassen sich bereits 30% der Fehler erkennen. Bei einer Blockgröße von 21 können über 50% der Fehler erkannt werden. Einmalig auftretende Fehler lassen sich damit mit nicht zuverlässig erkennen, da das Video jedoch aus mehreren Bildern besteht, ist es nicht unwahrscheinlich, dass ein solcher Fehler

wiederholt auf mehreren Bildern auftritt. Durch dieses wiederholte Auftreten steigt die Wahrscheinlichkeit, dass zumindest einer der sich wiederholenden Fehler erkannt werden kann.

Croppingfehler

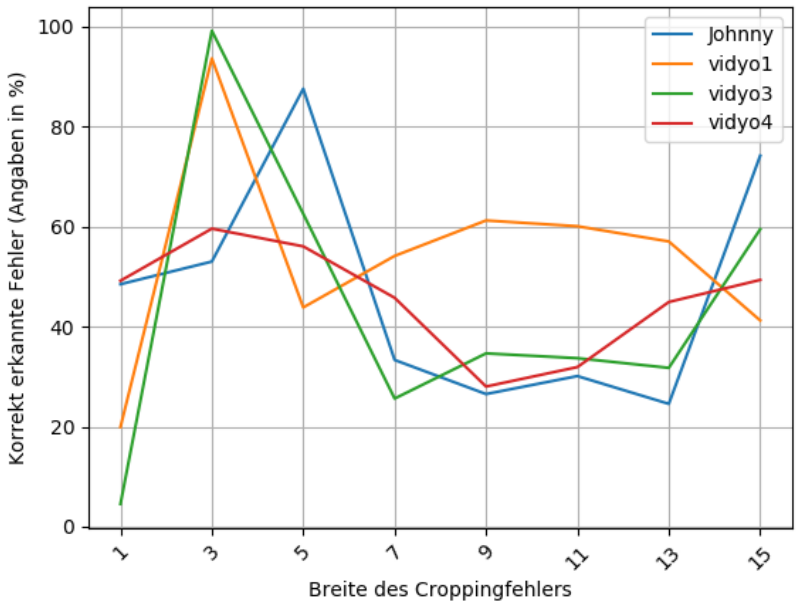
Im Folgenden wird die Erkennungsrate der Croppingfehler untersucht. Dabei wird ein schwarzer Balken, der das Abschneiden simuliert, einmal seitlich (links) und einmal oben eingebettet. Zusätzlich wurde die Breite der Croppingfehler variiert.

Abbildung 9.8 zeigt die Fehlererkennung für Croppingfehler bei unterschiedlichen Fehlerbreiten, einmal für oben eingefügte Fehler (Abbildung 9.8a) und einmal für seitlich eingefügte (Abbildung 9.8b). Dabei sind auf der Y-Achse die korrekt als fehlerhaft erkannten Blöcke aufgetragen. Abbildung 9.8a zeigt die als Fehler erkannten Blöcke bei Croppingfehlern am oberen Rand. Es fällt auf, dass die Fehlererkennung bei unterschiedlichen Videos signifikant variiert. Bei einigen Videos ließen sich bereits 50% der durch einen Pixel breite Croppingfehler betroffenen Blöcke korrekt als fehlerhaft erkennen. Bei anderen Videos wurden unter gleichen Gegebenheiten fast keine Fehler erkannt. Ähnlich stark variieren auch die Ergebnisse bei den meisten anderen Fehlerbreiten.
Abbildung 9.8b zeigt ein ähnlich ungleiches Bild, so werden bei den Videos „vidyo1“ und „vidyo3“ bei einer Croppingfehler-Breite von 5 Pixel nur noch ein sehr geringer Anteil an Fehlern erkannt, während beim Video „Johnny“ bei derselben Croppingfehler-Breite immer noch nahezu 100% der durch Croppingfehler betroffenen Blöcke als fehlerhaft erkannt werden. Generell schwankt die Fehlererkennung stark und scheint in hohem Maße abhängig vom Bildinhalt zu sein. Sowohl bei den seitlichen, als auch bei den oben eingefügten Fehlern konnten für einige Fehlerbreiten und einige Videos hohe Fehlererkennungsraten erreicht werden. Bei anderen Videos ließen sich unter gleichen Voraussetzungen jedoch keine Fehler erkennen. Eine einzelne eindeutige Erklärung für die Schwankung war trotz gründlicher Untersuchungen nicht zu finden.

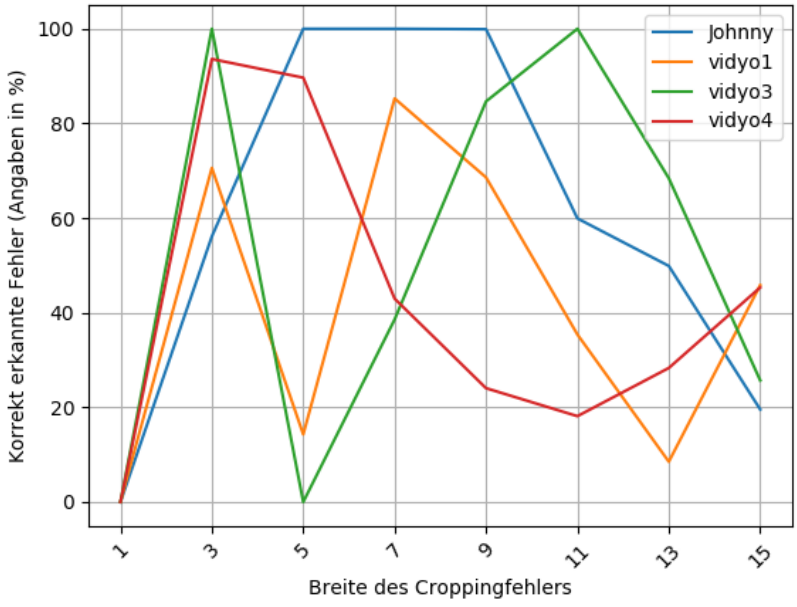
Das zumindest für Breiten unter 3 Pixeln die Fehlererkennung bei fast allen Videos zunimmt (auch wenn die konkrete Zunahme variiert) und bei unterschiedlichen Breiten wieder absinkt, lässt sich mit dem bereits in Kapitel 8 beschriebenen Überspringen von Buckets erklären. Dabei kann es vorkommen, dass die Summe so weit verschoben wird, dass sie nicht wie vorgesehen in den nächsten Bucket verschoben wird, sondern in den übernächsten. Da dieser Bucket, durch die alternierende Anordnung der Buckets wieder denselben Wert kodiert, wie der ursprüngliche, wird die Veränderung vom System nicht als Fehler erkannt. Dass die Verschiebung trotz relativ kleiner Fehler hier so starke Auswirkungen hat, liegt an der Form der Gewichtungsfunktion. Durch die Gewichtungsfunktion werden die niedrigen Frequenzen stärker gewichtet, als die hohen. Das Einfügen eins Croppingfehlers verändert nun je nach dem, wo er eingebettet wird, die Frequenzen, die sich horizontal bzw. vertikal ausbilden. Dies beinhaltet immer auch einige niedrige Frequenzen.

Dass vor allem beim Video „Johnny“ die erkannten Fehler von den anderen variieren, indem erst bei höheren Croppingfehler-Breiten das Maximum der erkannten Fehler erreicht wird, als es bei den anderen Videos der Fall ist, legt nahe, dass die Erkennung mit den bereits vorher im Bild bestehenden Strukturen zu erklären ist. So hat das Video „Johnny“ auf der linken Seite wie in Abschnitt 9.1.1 dargestellt, bereits relativ dunkle Bereiche. Dadurch sind schon einige Strukturen vorhanden, die sich in den DCT-transformierten Bilddaten ähnlich ausbilden, wie es Croppingfehler tun.

Generell kann festgehalten werden, dass Croppingfehler nicht immer zuverlässig für jeden Videoinhalt erkennbar sind. Durch den Einsatz mehrerer Videos steigt jedoch die Wahrscheinlichkeit, Fehler zu erkennen, stark an. Hinzu kommt, dass, um das Auftreten eines Fehlers zu erkennen, nicht alle, durch das Cropping veränderten Blöcke als Fehler erkannt



(a) Croppingfehler am oberen Rand



(b) Croppingfehler am linken Rand

Abbildung 9.8: Die Croppingfehler, die korrekt erkannt wurden.

werden müssen. Einige wenige erkannte Fehler würden ausreichen, um eine weitergehende händische Analyse der Daten zu rechtfertigen.

9.2.2 Zeitliche Fehler

Nachfolgend soll untersucht werden, wie zuverlässig das System die Fehler Jitter und Delay erkennen kann. Wie im Kapitel 6.3 beschrieben, werden die zeitlichen Zusammenhänge der Bilder als Zusatzinformationen in die Einzelbilder eingebettet. Damit hängt die Erkennung der zeitlichen Fehler direkt von der Robustheit des Wasserzeichens ab. Die Robustheit wurde bereits untersucht und wird daher in diesem Abschnitt nicht noch einmal behandelt.

Für die Analyse wurde ein Ausschnitt der in der Einleitung beschriebenen Videos verwendet. In real existierenden Szenarios können Jitter und Delay in Kombination auftreten. In dieser Arbeit wurde ein Konzept vorgestellt, welches auf das generelle Erkennen von zeitlichen Fehlern spezialisiert ist. Das Trennen von gleichzeitig auftretendem Jitter und Delay wurde daher nicht vorgenommen und die beiden Fehlerarten wurden streng voneinander getrennt betrachtet. Im Rahmen dieser Arbeit wurden sowohl Jitter als auch Delay lediglich simuliert, dafür wurden die Zeitstempel beim Einbetten und Extrahieren manipuliert. Eine genaue Beschreibung, wie dies implementiert wurde, ist in Kapitel 7.2.2 zu finden.

Delay

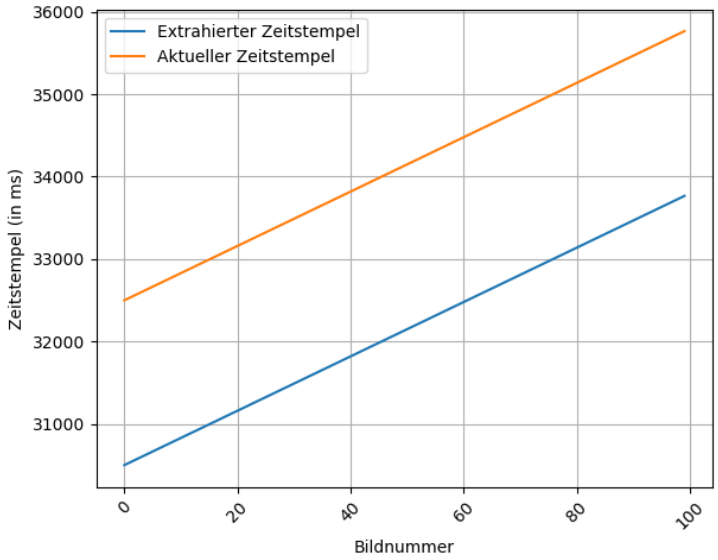
Im folgenden Abschnitt soll die Erkennung von Delay-Fehlern erklärt werden. Für das hier gegebene Beispiel wurde ein gleichmäßiges Delay von 2000 Millisekunden simuliert. Abbildung 9.9 zeigt den eingebetteten Zeitstempel im Vergleich zum Zeitstempel zum des Extraktionszeitpunktes. Die beiden liegen wie erwartet parallel zueinander. Das bedeutet, dass beim Extrahieren ein gleichbleibendes Delay erkannt werden konnte. Abbildung 9.9b zeigt die errechnete Differenz zwischen dem eingebetteten Zeitstempel und dem Zeitstempel des Extraktionszeitpunktes. Es zeigt, dass das simulierte Delay von 2 Sekunden erkannt werden konnte.

Jitter

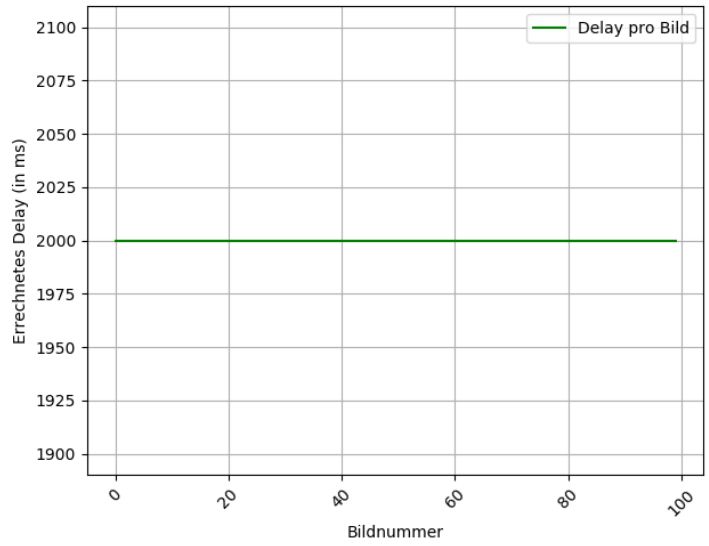
Jitter bedeutet, dass die Zeitspanne zwischen dem Einbetten und dem Extrahieren bei einzelnen Frames stark variiert. Eine genaue Definition ist in Kapitel 5.2.2 zu finden.

Um Jitter zu detektieren, wurde ein Zeitstempel während der Wasserzeicheneinbettung generiert und in das Wasserzeichen eingebettet. Das Prüfsystem kann, wie in Kapitel 6.3 beschrieben, zum Zeitpunkt der Extraktion den momentanen Zeitstempel erzeugen. Durch Errechnen der Differenzen lässt sich das Delay dann bildgenau detektieren. Variieren die Delays der einzelnen Bilder stark voneinander, liegt ein Jitter-Fehler vor.

In Abbildung 9.10 sind der vom System eingebettete (blau) und der erwartete (orange) zeitliche Verlauf dargestellt. Die Abweichung der eingebetteten Werte vom der erwarteten Zeit, stellt bildgenau den Delay dar.
Der Algorithmus konnte den Jitter hierbei bildgenau darstellen und auch die variierenden Unterschiede in den Delays sind in der Kurve klar zu erkennen. Damit lässt sich Jitter zuverlässig feststellen. Voraussetzung hierfür ist allerdings, dass das Wasserzeichen fehlerfrei extrahiert wird bzw. durch den fehlerkorrigierenden Code wiederhergestellt werden kann.



(a) Der extrahierte Zeitstempel im Vergleich zum momentanen



(b) Das errechnete Delay zwischen Einbettungszeitpunkt und Extraktionszeitpunkt

Abbildung 9.9: (a) Der extrahierte Zeitstempel in Abhängigkeit von der Bildnummer. (b) Die errechnete Verzögerung pro Bild.

9.2.3 Bitstromfehler

Zur Untersuchung von der in Kapitel 5.2.2 definierten Bitstromfehler wurde ein künstlicher Paketverlust simuliert. Dabei werden nacheinander NAL-Units im Video gelöscht. Die genaue Umsetzung lässt sich in Kapitel 7.2.3 finden.

Dabei sollte beachtet werden, dass es sich bei den ersten NAL-Units, häufig um „Picture parameter set“ oder IDR-Frames[3, p. 102] handelt. Diese können nicht gelöscht werden, da beide vom Decoder benötigt werden und sich das Video ohne diese nicht decodieren lässt.

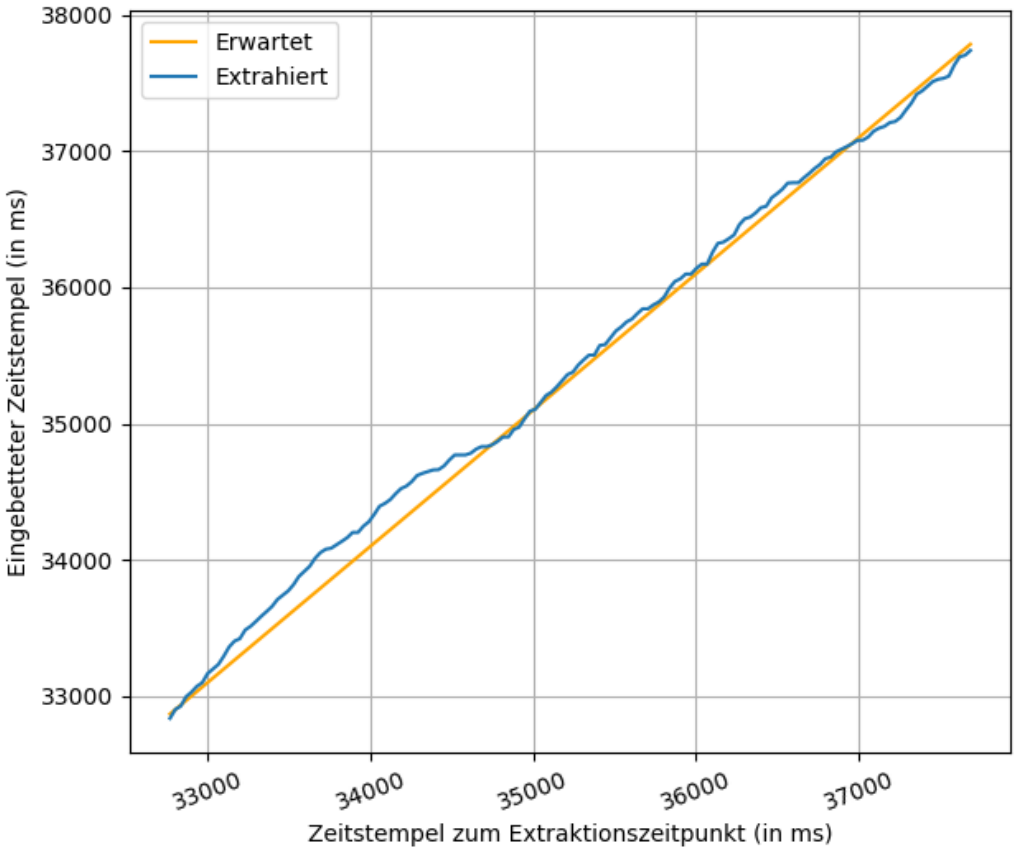


Abbildung 9.10: Der Zeitstempel zum Extraktionszeitpunkt im Verhältnis zum eingebetteten Zeitstempel erlaubt den Rückschluss auf die Stärke des Jitters. Ohne Jitter wären in dieser Abbildung zwei perfekte Geraden zu sehen.

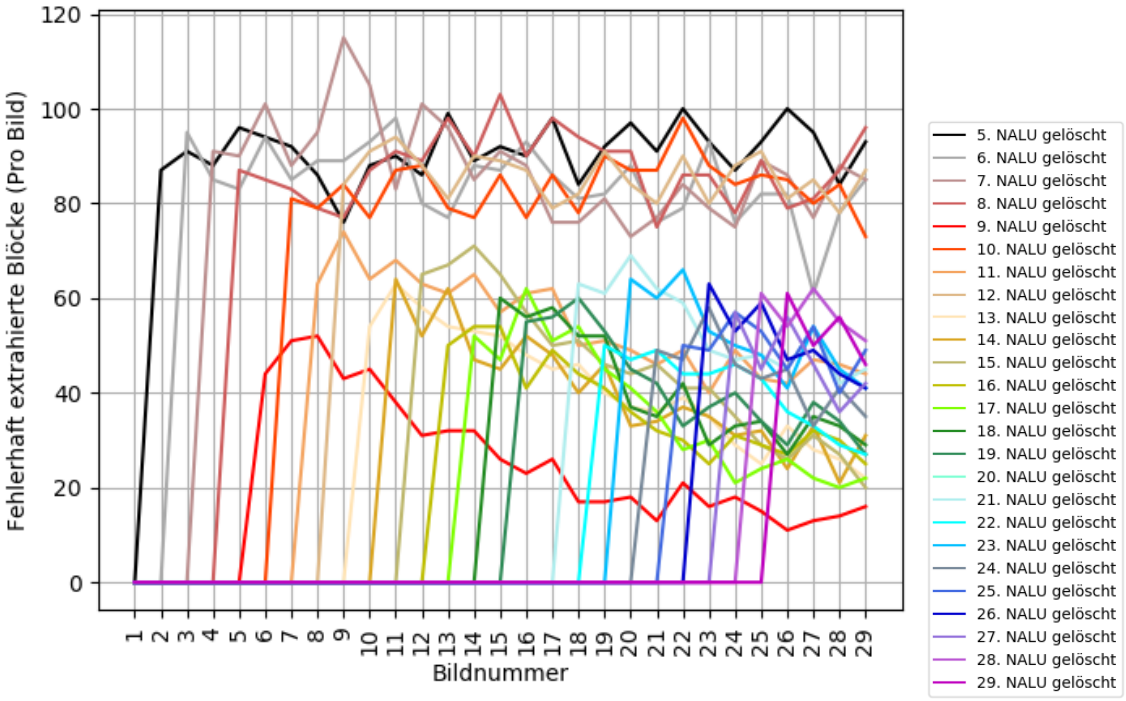


Abbildung 9.11: Die Anzahl der erkannten Fehler nach Löschen einer NAL-Unit.

Abbildung 9.11[1] zeigt 30 Bilder des Videos „Johnny“, wobei sukzessiv aufsteigend einzelne NAL-Units gelöscht wurden, angefangen bei der ersten Non-IDR[3, p. 102] NAL-Unit. Non-IDR-Bilder kodieren nur den Unterschied zum vorhergehenden Bild, wie jedoch im Abschnitt 9.1.1 herausgestellt, wird durch das Wasserzeichen neue Bewegung ins Bild gebracht, welche dann in den Non-IDR-Bildern kodiert wird. Durch die zusätzliche Bewegung müssen auch mehr Veränderungsinformationen in einem Non-IDR-Bild codiert werden. Durch das Löschen eines solchen Non-IDR-Bildes treten auch stärkere Veränderungen am Bild in Erscheinung. Abbildung 9.12 illustriert, wie sehr das Wasserzeichen vom Fehlen einer einzelnen NAL-Unit betroffen ist. Während der ursprüngliche Bildinhalt relativ intakt geblieben ist, da sich der Moderator im Video „Johnny“ relativ wenig bewegt hat, sind die Blockartefakte, die durch das Wasserzeichen eingebracht wurden, im unteren Teil des Bildes stark verändert. Durch die fehlende NAL-Unit wird immer auch ein Teil des Videos gelöscht. Damit wirkt sich ein solcher Fehler immer auch auf alle nachfolgenden Bilder aus, was auch in Abbildung 9.11 zu erkennen ist. Zudem ist der Abbildung zu entnehmen, dass in den allermeisten Fällen weitaus mehr Bits verändert werden, als durch den gewählten BCH-Code korrigierbar sind. Damit ist auch die Bildidentifikations-Information in allen Folgebildern nicht mehr fehlerfrei zu extrahieren. Alle auf eine fehlende NAL-Unit folgenden Bilder sind also nicht mehr eindeutig den abgesendeten Bildern zuzuordnen. Um trotzdem eine Aussage treffen zu können, wie viele Fehler erkannt wurden, wurden zur Visualisierung von Abbildung 9.11 zufällige aber nachvollziehbare Werte eingebettet. Die Synchronisierung wurde manuell vorgenommen, indem dem Prüfsystem die gelöschte NAL-Unit bekannt war. Das System hat anschließend die fehlende NAL-Unit als fehlendes Bild behandelt und den Abgleich entsprechend angepasst, sodass das fehlende Bild für den Algorithmus als nicht existent behandelt wurde.

Zudem zeigt Abbildung 9.11, dass es vorkommen kann, dass der Decoder Probleme hat das Video wieder korrekt zu decodieren. So zeigt Abbildung 9.11, dass beim gegebenen Video das Löschen der 20. Non-IDR NAL-Unit dazu geführt hat, dass das Decodieren des Videos nach dem

[1] Da hier die Fehlerkorrektur von 30 Bits nicht mehr ausreichend war, wurden hier zufällige binäre Codewörter verwendet, die mithilfe der Funktion „np.random.randint“ erstellt wurden. Als „Seed“ wurde die aktuelle Bildnummer verwendet.

Abbildung 9.12: Ein Beispiel für einen Bitstromfehler.

16. Bild mit einer Fehlermeldung abgebrochen wurde. Theoretisch ist denkbar, dass es zu Paketverlust kommt und damit einzelne Bilder fehlen, die darauffolgenden Bilder jedoch korrekt decodiert werden können. Dies könnte beispielsweise der Fall sein, wenn das Video komplett aus I-Frames besteht oder das nächste ankommende Bild zufällig ein I-Frame ist. Dies ließe sich über die eingebettete Bildnummer erkennen.

9.3 VISUELLER EINFLUSS

In diesem Abschnitt wird analysiert, inwiefern das Wasserzeichen einen visuellen Einfluss auf das Bild hat und wie dieser Fehler von einem Betrachter wahrgenommen werden würde.

Im Rahmen dieser Arbeit lag der Fokus nicht auf dem visuellen Einfluss, da die Entwicklung eines automatisierten Testsystems im Vordergrund stand. Trotzdem soll überprüft werden, ob es sich bei den Videos weiterhin um natürliche Videos handelt, wie es in der Aufgabenstellung gefordert ist. Zudem ist der Fall denkbar, dass ein solches Testsystem durch einen Fehler, wie beispielsweise menschliches Versagen, in einem Produktiv-System aktiviert wird. Dadurch würde das Wasserzeichen auch die Bilder im Livebetrieb beeinflussen. Auch wenn dies eher als unwahrscheinlich einzustufen ist, soll untersucht werden, wie störend sich ein solches Testsystem auf den Betrachter auswirken würde.

Zur Beurteilung wie stark das Bild durch das Wasserzeichen verändert wird, wird zum einen die „Peak signal-to-noise ratio“ (PSNR) als objektive Metrik herangezogen, um die Intensität der Veränderung feststellen zu können. Zum anderen werden Testpersonen befragt, wie stark die Veränderungen subjektiv stören. Zusätzlich wird evaluiert, ob das Bild von den Probanden trotz des Wasserzeichens noch gut erkennbar ist.

9.3.1 Objektiver Eindruck

Um einen objektiven Eindruck des Einflusses zu erhalten, wird hier der PSNR verwendet. Der PSNR ist in Kapitel 3.1 erklärt. Dabei wird zuerst der PSNR zwischen dem Originalbild und dem durch das Wasserzeichen veränderte Bild berechnet. Anschließend wurde der Einfluss des Wasserzeichens mit dem Einfluss der verlustbehafteten Kompression verglichen.

QP	PSNR ohne Wasserzeichen	PSNR mit Wasserzeichen
20	44.679047	23.9143719
20	44.679047	23.9143719
30	40.097415	23.868200
40	34.822502	23.711452

Tabelle 9.1: Der visuelle Einfluss gemessen mit der objektiven Metrik des PSNR. Berechnet wurde der Unterschied des unveränderten Videos zum komprimierten und der Unterschied des unveränderten Videos zum mit dem Wasserzeichen versehenen und komprimierten Video.

In Tabelle 9.1 sind die Ergebnisse der Untersuchung dargestellt. Dabei ist der PSNR zwischen dem originalen Bild und dem durch das Wasserzeichen veränderten Bild berechnet worden. Zudem wurde der PSNR zwischen Originalbild und dem mit unterschiedlichen Parametern encodierten Bild dargestellt.
Das Wasserzeichen verändert das Bild dabei weitaus stärker, als es die Kodierung mit dem x264 tut. Selbst starke verlustbehaftete Kompression wirkt sich weniger stark auf den PSNR aus, als die Veränderungen, die durch das Wasserzeichen eingebettet wurden. Objektiv betrachtet, wurde das Bild also stark verändert. Nachfolgend wird untersucht, ob sich diese Erkenntnis in subjektiven Tests bestätigen lässt.

9.3.2 Subjektiver Eindruck

In diesem Abschnitt soll der subjektive Eindruck überprüft werden, den das Wasserzeichen auf den Betrachter hat. Dafür wurden Befragungen von sieben Probanden durchgeführt, die ihren subjektiven Eindruck beurteilen sollten. Hierzu wurde den Probanden ein Fragebogen mit neun Fragen vorgelegt. Im Rahmen der Befragung wurden zwei Videos gezeigt, einmal das originale unveränderte Video und einmal das durch das Wasserzeichen veränderte Video. Der Inhalt des Videos bestand aus den im Anfang vom Kapitel 9 definierten vier Videos.

Der Fragebogen war dabei dreigeteilt. Die ersten zwei Fragen sollten den ersten Eindruck der Probanden erfassen und sollten bereits nach fünf Sekunden beantwortet werden. Nach längerem Betrachten wurden diese beiden Fragen erneut gestellt. Die restlichen Fragen durften erst nach Beenden des Videos bearbeitet werden. Dabei wurde die Möglichkeit geboten, wenn gewünscht, die zwei Videos vor Beantwortung der Fragen 5-9 noch einmal vollständig zu betrachten. Die Fragen waren anhand einer 5-teiligen Likert-Skala[23] zu beantworten. Die Antwortmöglichkeiten waren: „kann ich nicht beurteilen“, „trifft nicht zu“, „trifft eher zu“, „teils-teils“, „trifft weniger zu“ und „trifft voll zu“. Für die Auswertung wurden dabei die Antworten in gezeigter Reihenfolge von null anfangend, aufsteigend bis 5 gewichtet. Ein solcher Fragebogen befindet sich im Anhang.

Die Fragen lauteten wie folgt:

1. Das Bild ist wenig verändert. (nach 5 Sekunden)
2. Mir ist aufgefallen, dass die Artefakte sich im Hintergrund stärker ausbilden, als im Vordergrund. (nach 5 Sekunden)

3. Das Bild ist wenig verändert. (nach 30 Sekunden)

4. Mir ist aufgefallen, dass die Artefakte sich im Hintergrund stärker ausbilden, als im Vordergrund. (nach 30 Sekunden)

5. Der Inhalt des Videos ist weiterhin erkennbar.

6. Ich würde eine Video-Konferenzsoftware mit solcher Bildqualität weiterverwenden.

7. Unabhängig von Ihrer Antwort auf die vorhergehende Frage: Ich würde mich nach alternativer Video-Konferenzsoftware umsehen.

8. Ich wäre bereit, die Qualitätseinbußen für einzelne Sessions zu tolerieren, wenn ich wüsste, dass dadurch die Qualität für zukünftige Video-Konferenzen verbessert wird.

9. Die hier gezeigten Artefakte sind störender, als typische Kodierungsartefakte.

Bezeichung	1. Frage	2. Frage	3. Frage	4. Frage	5. Frage	6. Frage	7. Frage	8. Frage	9. Frage
Mittelwert	1,286	3,143	1,429	2,143	4,857	1,571	4,714	2,714	2,429
Median	1	4	1	2	5	1	5	3	2
Min.[2]	1	1	1	1	4	1	4	1	2
Max.	3	4	4	3	5	3	5	5	5
Std. Abw.	0,7	1,125	1,05	1,457	0,35	0,728	0,452	1,161	1,917
Varianz	0,571	1,265	1,102	2.122	0,122	0,531	0,204	1,357	3,673

Abbildung 9.13: Statistische Auswertung der Umfrage

Einige statistische Kennwerte zu den einzelnen Ergebnissen finden sich in Tabelle 9.13. Die genauen Antworten, aufgeschlüsselt nach Probanden, finden sich im Anhang A.5. Der Ersteindruck der Probanden ist, dass das Bild stark verändert wurde. Einige Probanden empfanden die Veränderungen nach längerer Betrachtung allerdings als weniger störend. Den meisten fallen zudem die im Hintergrund stark ausgeprägten Artefakte nach längerem Betrachten weniger stark auf.
Wie in Abschnitt 9.1.1 bereits erläutert erscheinen Artefakte in Bereichen mit mehr Struktur weniger auffällig. Die Probanden konzentrieren sich unter Umständen mit zunehmender Betrachtungslänge stärker auf die „interessanten" Bereiche des Bildes, wie zum Beispiel auf den Kopf des Moderators. Dadurch fallen ihnen die Artefakte im Hintergrund weniger stark auf.

Relativ eindeutig fällt die Einschätzung der Probanden aus, dass das Bild weiterhin gut erkennbar ist. Daher liegt es nahe, dass das Ziel der Aufgabenstellung ein natürliches Video zu verwenden, erfüllt wurde. Zwar ist der Bildinhalt verändert, wie es auch im Rahmen der Aufgabenstellung erlaubt ist, der Bildinhalt ist jedoch weiterhin vorhanden und erkennbar.

Die subjektive Einschätzung ergibt auch, dass falls dieses Testsystem einmal fälschlicherweise im Produktivbetrieb aktiviert werden sollte, dies von den Nutzern als störend und nicht akzeptabel angesehen werden würde. Nahezu alle Probanden würden ein Produkt mit einer solchen Videoqualität nicht weiterverwenden und sich nach Alternativprodukten umsehen. Die Antwort auf Frage 8 zeigt, dass ein eingeblendeter Hinweis, dass diese Videoqualität nur vorübergehend ist und für die Verbesserung der Videoqualität sorgen könnte, dazu führen könnte, die Ablehnung durch den Nutzer zu verringern.

Die Probanden sind eher uneinig darüber, ob die entstehenden Artefakte weniger störend sind als reguläre Kodierungsartefakte.

[2] Ohne die Antwortmöglichkeit "kann ich nicht beurteilen"

9.4 ZUSAMMENFASSUNG

In diesem Kapitel wurde dargestellt, welche Ergebnisse durch die entwickelte Technik erreicht werden konnten. Dabei wurde herausgefunden, dass die eingesetzten Techniken bei schwacher bis mittelstarker Kompression durch den Video-Codec robust sind. Bei starker Kompression ist die Robustheit nicht mehr für alle Bilder erreichbar und damit können einzelne Wasserzeichen nicht mehr fehlerfrei extrahiert werden.

Die Fehlererkennung fällt dabei sehr unterschiedlich aus. Bei der Bildfehlererkennung kann bei Blockfehlern lediglich eine Fehlererkennungsrate von etwas über 50% erreicht werden. Wohingegen bei Croppingfehlern teilweise eine Fehlererkennungsrate von über 100% erreicht wurde, allerdings schwankt die Croppingfehler-Detektion, abhängig vom Bildinhalt, sehr stark.

Zeitliche Fehler werden hingegen problemlos erkannt, sofern das Wasserzeichen korrekt extrahiert werden kann. So lassen sich sowohl Delay als auch Jitter anhand der eingebetteten Informationen zuverlässig detektieren.

Bitstromfehler, die sich als Fehler im Bild darstellen, sind in der Regel ebenfalls gut zu detektieren, da hier große Teile des Bildes stark verändert werden. Zwar ist es aufgrund des eingesetzten zyklischen Fehlercodes möglich, dass durch eine sehr hohe Anzahl an fehlerhaft extrahierten Wasserzeichen-Bits wiederum ein gültiges Codewort entsteht; es ist jedoch nahezu ausgeschlossen, dass die extrahierten Zeitstempel und Bildnummern per Zufall den erwarteten Informationen entsprechen, wodurch auch in diesem Fall eine robuste Detektion des Bitstromfehlers möglich ist.

Sowohl die objektive Metrik, als auch die subjektiven Ergebnisse zeigen, dass das Bild stark verändert wurde. Dabei gibt die objektive Metrik an, dass das Bild weitaus stärker verändert wird, als selbst durch stark verlustbehaftete Kompression. Auch die befragten Probanden gaben relativ eindeutig an, dass das Wasserzeichen als störend wahrgenommen wird. Die meisten würden ein Videosystem mit einer solchen Videoqualität nicht weiterverwenden. Es wird jedoch ebenfalls von einer Mehrheit angegeben, dass der Bildinhalt weiterhin gut erkennbar ist. Daraus ergibt sich, dass die Veränderung zwar relativ stark ist, es sich aber weiterhin um natürliche Bildinhalte handelt, wie in der Aufgabenstellung gefordert. In Tabelle 9.2 ist dargestellt, welche der am Ende von Kapitel 5 definierten Ziele erfüllt werden konnten. Ein Haken ✓ steht dabei für ein uneingeschränktes Erreichen des Ziels, ein Kreuz ✗ für das Nichterreichen eines Ziels und ein Haken in Klammern für ein teilweise erreichtes Ziel.

ID	Name	Erfüllt	Einschränkung
		Allgemeine Ziele	
A1	Robustheit	(✓)	Eine Robustheit des Wasserzeichens gegenüber dem Encoder konnte nur bis zu einem QP von 28 erreicht werden.
		Fehlererkennung	
P1	Croppingfehler	(✓)	Croppingfehler können nicht immer zuverlässig erkannt werden, so konnte eine Fehlererkennung von bis zu 100% erreicht werden, allerdings hängt die Erkennung stark vom Bildinhalt ab.
P2	Blockfehler	(✓)	Blockfehler konnten abhängig von der Blockgröße in bis zu 50% der Fälle erkannt werden.
T1	Jitter	✓	
T2	Delay	✓	
B1	Paketverlust	✓	Bitstromfehler können relativ zuverlässig erkannt werden. Es besteht jedoch die Gefahr, dass bei einer zu hohen Anzahl an Fehlern der BCH-Code ein gültiges Codewort rekonstruiert, das sich von dem ursprünglichen Codewort unterscheidet. Die extrahierten Informationen entsprechen dennoch nicht den erwarteten Werten (z.B. aufsteigende Bildnummer und Zeitstempel), wodurch sich dies dennoch als Fehler detektieren lässt.
		Visueller Einfluss	
V1	Natürliches Video	✓	Der Bildinhalt ist stark verändert. Der Bildinhalt der Eingabevideos ist jedoch weiterhin gut erkennbar.

Tabelle 9.2: Status der Ziele und eventuelle Einschränkungen

10 FAZIT UND AUSBLICK

FAZIT UND AUSBLICK

Im folgenden Kapitel werden die Erkenntnisse und Ergebnisse aus den vorhergehenden Kapiteln noch einmal zusammengefasst, es wird ein Fazit gezogen und es werden einige interessante Forschungsansätze für zukünftige Projekte kurz beschrieben.

10.1 ZUSAMMENFASSUNG

In dieser Arbeit wurde ein Algorithmus entwickelt und prototypisch umgesetzt, der es erlaubt, ein Videoverarbeitungssystem auf Fehler zu testen. Das Videoverarbeitungssystem besteht dabei aus einer Encoder-Decoder-Verarbeitungskette, welche als Blackbox betrachtet werden soll. Die Fehlererkennung soll somit ausschließlich anhand der Ausgabedaten erfolgen. Dabei wurden drei Kategorien von zu erkennenden Fehlern definiert: Bildinhaltsfehler, zeitliche Fehler und Bitstromfehler. Aus den Anforderungen, die an das System gestellt wurden, ergaben sich drei übergreifende Ziele: die Robustheit, die Fehlererkennung und der visuelle Einfluss. Die Robustheit wurde so definiert, dass das System auch dann problemlos arbeiten kann, wenn die Daten durch die verlustbehaftete Kompression des Encoders verändert wurden. Die Fehlererkennung definiert, dass die drei Fehlerkategorien erkannt werden können. Das letzte Ziel besagt, dass der visuelle Einfluss der eingesetzten Techniken minimiert werden soll, sodass weiterhin natürliche Bilddaten in den Videos Verwendung finden.

Um eine gegen verlustbehaftete Kompression robuste Fehlererkennung zu ermöglichen, wurde das System durch den Einsatz eines Wasserzeichens realisiert. Aufbauend auf dem Ansatz der semi-fragilen Wasserzeichen, die schon in anderen Arbeiten dazu benutzt wurden, Manipulationen in Bildern zu erkennen, wurde ein neues semi-fragiles Wasserzeichen entwickelt. Angelehnt an das Prinzip von QR-Codes, erkennt dieses Wasserzeichen Fehler in den eingebetteten Informationen mithilfe der Redundanzen und kann diese korrigieren. Dabei wurden zwei getrennte Module entwickelt, ein Einbettungssystem und ein Prüfsystem. Das Einbettungssystem bettet das Wasserzeichen vor der Verarbeitung durch das Videoverarbeitungssystem in die unkomprimierten Videodaten ein. Das Prüfsystem extrahiert und überprüft das Wasserzeichen nach der Verarbeitung.

Um zeitliche Fehler, wie Verzögerungen einzelner Bilder oder des gesamten Videos, zu erkennen, wurden in die einzelnen Bilder blockweise zeitliche Informationen eingebettet und durch einen fehlerkorrigierenden BCH-Code gegen Veränderungen abgesichert. Die semi-fragile Eigenschaft des Wasserzeichens wurde dabei zur Fehlererkennung verwendet. Dabei werden die fragilen Eigenschaften des Wasserzeichens zur Erkennung von Fehlern, die im Bild entstehen, angewendet. Die robusten Eigenschaften des Wasserzeichens werden dazu eingesetzt, diese Fehlererkennung auch dann zu gewährleisten, wenn das Bild durch verlustbehaftete Kompression verändert wurde. Der Algorithmus arbeitet dabei zur Erhöhung der Stabilität und zur Reduktion des visuellen Einflusses auf den durch die Diskrete Kosinustransformation (DCT) transformierten Werten im Frequenzbereich. Dafür werden die Bilddaten in Blöcke aufgeteilt, die anschließend durch die DCT transformiert werden. Dabei wird die Summe der DCT-Koeffizienten je nach einzubettendem Wert nach unten bzw. nach oben verschoben. Das Verschieben geschieht unter Zuhilfenahme einer Gewichtungsfunktion. Der Wertebereich wurde hierfür in mehrere Teilbereiche eingeteilt. Anschließend wurde der Einfluss verschiedener Parameter auf die zu erreichenden Ziele – Robustheit, Fehlererkennung und visueller Einfluss – hin analysiert. Dabei wurde eine Gewichtungsfunktion entwickelt, die solche DCT-Koeffizienten besonders stark gewichtet, bei denen die Veränderungen durch verlustbehaftete Kompression gering ausfallen.

Durch das so entwickelte System ließ sich eine Robustheit gegenüber geringen und mittleren

Kompressionsraten erreichen. Zudem konnte eine Fehlererkennung von bis zu 100% der Fehler auf Bildebene, sowie eine hundertprozentige Fehlererkennung von zeitlichen Fehlern erzielt werden. Die Erkennung der Bildfehler variierte jedoch sehr stark abhängig vom Bildinhalt und den eingebetteten Fehlern. Die Bitstromfehler konnten selbst bei geringem Paketverlust erkannt werden und lassen sich zuverlässig erkennen. Eine Analyse des visuellen Einflusses hat ergeben, dass das Wasserzeichen von vielen Probanden als störend wahrgenommen wird, der Bildinhalt jedoch trotzdem weiterhin erkennbar bleibt.

10.2 FAZIT

Im Rahmen dieser Arbeit wurde ein semi-fragiles Wasserzeichen entwickelt, welches eine Videoverarbeitungskette anhand der Ausgabedaten automatisiert auf Fehler testen kann. Generell kann das Konzept noch weiter untersucht werden. Dabei könnte sicherlich noch ein interessantes Themengebiet sein, den visuellen Einfluss in Kombination mit Robustheit ohne Fehlererkennungseigenschaften des Wasserzeichens zu untersuchen.
Die Veränderungen, die durch die verlustbehaftete Kompression entstehen, variieren je nach Inhalt des betroffenen Bildes so stark, dass einige Bildfehler durch das vorgestellte Konzept nur schwer von Kompressionsartefakten zu unterscheiden sind. Dass diese beiden Ziele gleichzeitig so optimiert werden können, dass man eine hundertprozentige Bildfehlererkennung erreicht, ist daher eher unwahrscheinlich.

Die entwickelte Technik kann Fehler bildgenau detektieren. Da ein Video aus mehreren aufeinanderfolgenden Einzelbildern besteht, ist es möglich, dass ein auf einem Bild aufgetretener Fehler sich auf nachfolgenden Bildern wiederholt. Für viele Fehler ist eine hundertprozentige Bildfehlererkennung damit oftmals nicht notwendig. Mit jeder Wiederholung steigt die Wahrscheinlichkeit, dass diese Fehler erkannt werden. Damit kann das Konzept auch ohne hundertprozentige Erkennungsrate in vielen Fällen helfen, Fehler automatisiert zu erkennen.

Um den visuellen Einfluss des Wasserzeichens zu klassifizieren, wurden sowohl objektive als auch subjektive Tests durchgeführt. Der objektive Test hat dabei ergeben, dass das Bild durch das Wasserzeichen stark verändert wurde. Der subjektive Eindruck der befragten Probanden ergab jedoch, dass der Bildinhalt dennoch weiterhin gut erkennbar ist. Damit bestätigt sich, dass der auf Wasserzeichen basierende Ansatz zum Test von Videoverarbeitungssystemen mit natürlichen Videos erreicht wurde. Da der visuelle Eindruck von den meisten Probanden als störend wahrgenommen wird, was auch die objektive Metrik bestätigt, sollte der Einsatz des Wasserzeichens auf die Anwendung in Testumgebungen beschränkt bleiben.

10.3 AUSBLICK

Im Rahmen dieser Arbeit wurden Techniken zum automatisierten Testen einer Videoverarbeitungskette vorgestellt. Im folgendem sollen noch einige vielversprechende Ansätze für weitere Untersuchungen präsentiert werden.

Zukünftig könnte noch weiterer Aufwand in die Untersuchung einer sinnvollen Gewichtung investiert werden. Im Analysekapitel in Abschnitt 8.1.2 wurde eine Gewichtung vorgestellt, die die Frequenzen proportional zu ihren mittleren Werten gewichtet. Das exponentielle Wachstum, das dieser Gewichtung zugrunde liegt, wird in der Realität jedoch noch überschritten. Weitere Optimierung dieser grundlegenden Funktion würde eventuell noch zu einer höheren Robustheit führen.

Im Verlauf der Arbeit hat sich ergeben, dass die in Kapitel 6.2.2 vorgestellte Formel 6.1 zur Reduktion der Verschiebedistanz nicht immer ideale Ergebnisse liefert und dass es vorkommen kann, dass die Verschiebung nicht in den nächsten Bucket geschieht. Die Formel muss dabei angepasst werden, um alle theoretisch denkbaren Fälle bestmöglich abzufangen.

Eine Größenveränderung des Videos im Rahmen der Verarbeitungskette könnte sich als schwierig erweisen, da bei einer Vergrößerung bzw. Verkleinerung des Bildes teilweise signifikante Teile der Information verloren gehen. So geht eine Verkleinerung des Videos um den Faktor 2 mit einem Informationsverlust von 75% einher. Eventuell könnte die Gewichtung so variiert werden, dass die wegfallenden Informationen bereits beim Einbetten nicht in die Summenbildung einbezogen werden. Dies wäre ein interessanter Ansatzpunkt für weitere Untersuchungen.

Im Abschnitt 7.1.1 wurde erklärt, dass der Prototyp im Falle von Auflösungen, die in horizontaler bzw. vertikaler Richtung nicht problemlos durch die Blockgröße teilbar sind, die überstehenden Teile des Bildes ignoriert. Um in zukünftigen Arbeiten zu erlauben, dass das gesamte Bild verwendet wird, wäre es möglich, das Bild künstlich zu vergrößern, sodass die entstehende Auflösung ohne Rest durch die Blockgröße teilbar ist. Denkbar wäre hierbei einerseits, einen schwarzen Bereich einzufügen, was sich aufgrund der fehlenden Struktur als schwierig erweisen könnte. Alternativ wäre eine Vergrößerung der Bilddaten des Blocks durch Kopieren oder Spiegeln möglich, um somit Struktur zu erhalten.
Ein weiterer Ansatz wäre es, in besagten Randbereichen, die nicht mehr groß genug für die eigentliche Blockgröße sind, kleinere Blockgrößen zu verwenden.

Zusätzlich wäre es interessant, die Daten in alle drei Kanäle einzubetten und nicht nur in den Luminanzkanal, wie es hier getan wurde. Da sich viele Fehler auf alle drei Kanäle auswirken, könnten alle Kanäle getrennt voneinander betrachtet werden. Anschließend könnten dann die Ergebnisse, die zeigen, ob ein Fehler aufgetreten ist, zwischen den drei Kanälen abgeglichen werden. So lässt sich eventuell besser feststellen, ob ein Fehler aufgetreten ist oder ob die verlustbehaftete Kompression zu diesen Veränderungen geführt hat.

A ANHANG

A.1 VERÄNDERUNG DER DCT-KOEFFIZIENTEN DURCH VERLUSTBEHAFTETE KOMPRESSION

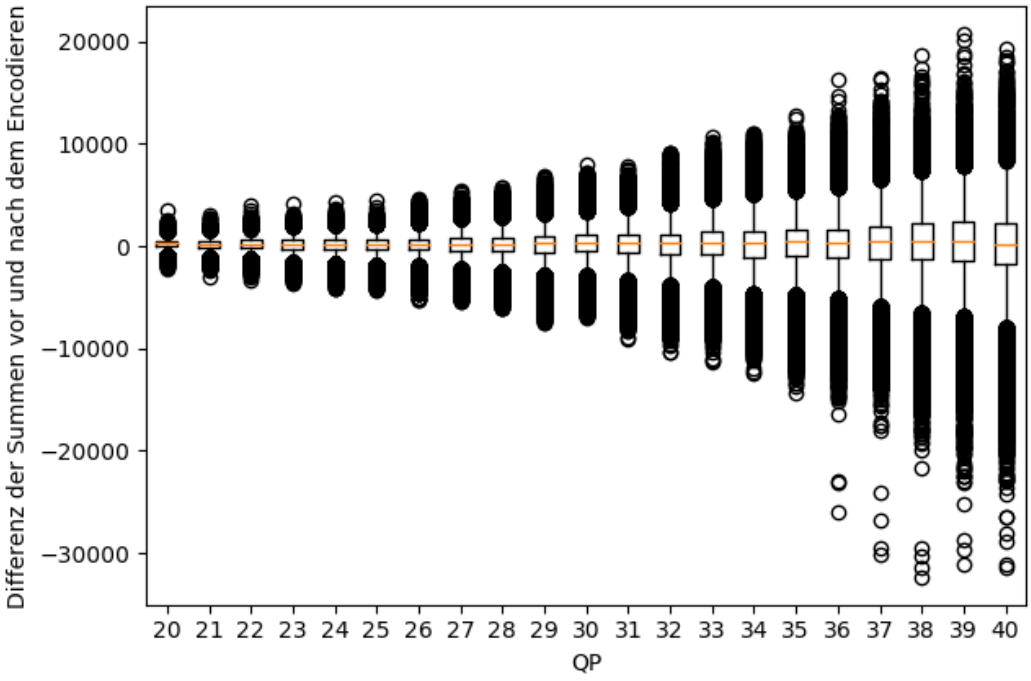


Abbildung A.1: Die Veränderungen der Kompression bei unterschiedlichen QPs

Abbildung A.2: Ausschnitt aus dem mit dem Wasserzeichen versehenem Video: vidyo1

A.2 AUSSCHNITTHAFTE DARSTELLUNG AUS DEM VIDEO MIT EINGEBETTETEM WASSERZEICHEN

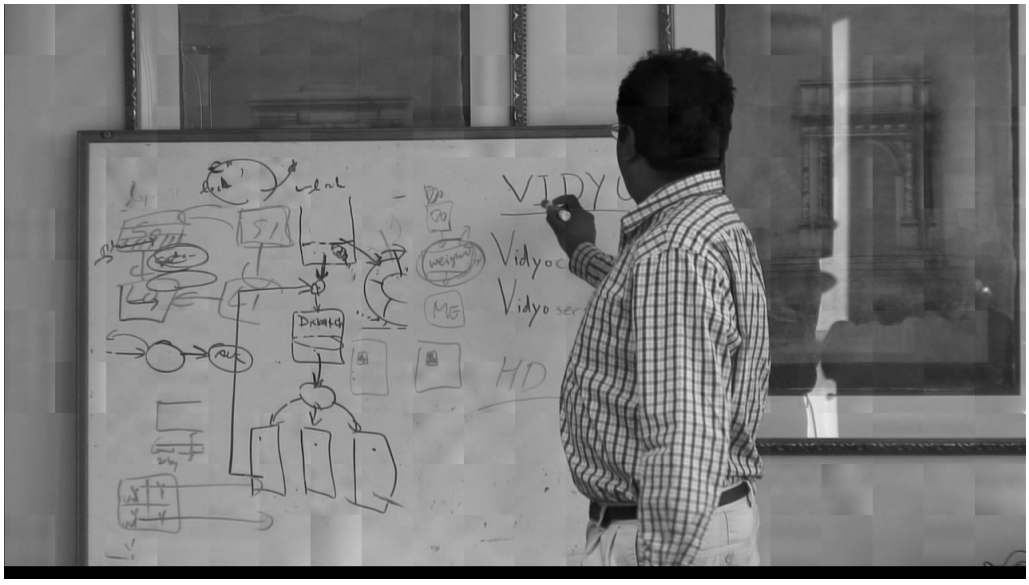


Abbildung A.3: Ausschnitt aus dem mit dem Wasserzeichen versehenem Video: vidyo3

A.3 FRAGEBOGEN

Fragebogen

Zur Masterarbeit: „Automatisiertes Ende-Zu-Ende-Video-Verifikationssystem für verteilte Video-Konferenzsysteme“.

Vielen Dank, dass Sie an dieser kurzen Meinungsumfrage teilnehmen. Ich möchte Sie bitten, den Fragebogen wahrheitsgemäß und nach eigenem Ermessen auszufüllen, hierbei gibt es keine richtigen oder falschen Antworten. Der Test dauert in etwa 10 Minuten und wird anonym durchgeführt. Dabei werden Ihnen zwei Schwarz-Weiß-Videos ohne Sound vorgespielt. Zuerst wird Ihnen das originale unveränderte Video vorgespielt, anschließend erhalten Sie ein verändertes Video. Dabei wird das veränderte Video nach wenigen Sekunden gestoppt und Sie sollen Ihren ersten Eindruck (2 Fragen) bewerten. Es soll herausgefunden werden, wie störend die Änderungen am Video auf den Betrachter wirken. Sie sollen sich dabei vorstellen, dass ein Video mit solcher Qualität bei Ihnen im Rahmen einer Video-Konferenz ankommt. Dabei soll weder eine Rolle spielen, dass das Video schwarz-weiß ist, noch, dass kein Sound abgespielt wird. Lediglich Ihre Empfindungen beim Anschauen des Videos sollen berücksichtigt werden. Auf der 2. Seite sind 2 Fragen, die Sie jeweils kurz nach Beginn des Videos beantworten sollen und noch einmal nach der Hälfte des Videos. Die Fragen der dritten Seite sollen erst nach Beenden des Videos ausgefüllt werden. Bei Interesse an den Ergebnissen geben Sie bitte auf dem gesondert gereichten Blatt Ihren Namen und Ihre E-Mail-Adresse an.

Der Fragebogen enthält Fragen nach dem folgenden Muster.

Bitte kreuzen sie das am ehesten Zutreffende an.

Beispielfrage:

1.) Die Netzabdeckung meines Mobilfunkanbieters ist gut:

☐	☐	☐	☐	☐	☐
trifft voll zu	trifft eher zu	teils-teils	trifft weniger zu	trifft nicht zu	kann ich nicht beurteilen

Erst beim Betrachten des zweiten Videos beantworten

Bitte bewerten Sie folgende Aussagen:

Erster Eindruck nach 5 Sekunden

1.) Das Bild ist wenig verändert:

☐	☐	☐	☐	☐	☐
trifft voll zu	trifft eher zu	teils-teils	trifft weniger zu	trifft nicht zu	kann ich nicht beurteilen

---

2.) Mir ist aufgefallen, dass die Artefakte sich im Hintergrund stärker ausbilden als im Vordergrund:

☐	☐	☐	☐	☐	☐
trifft voll zu	trifft eher zu	teils-teils	trifft weniger zu	trifft nicht zu	kann ich nicht beurteilen

---

Zweiter Eindruck nach 30 Sekunden

3.) Das Bild ist wenig verändert:

☐	☐	☐	☐	☐	☐
trifft voll zu	trifft eher zu	teils-teils	trifft weniger zu	trifft nicht zu	kann ich nicht beurteilen

4.) Mir ist aufgefallen, dass die Artefakte sich im Hintergrund stärker ausbilden als im Vordergrund:

☐	☐	☐	☐	☐	☐
trifft voll zu	trifft eher zu	teils-teils	trifft weniger zu	trifft nicht zu	kann ich nicht beurteilen

---

ERST NACH BEENDEN DES ZWEITEN VIDEOS AUSFÜLLEN

Bitte bewerten Sie folgende Aussagen:

5.) Der Inhalt des Videos ist weiterhin erkennbar:

☐	☐	☐	☐	☐	☐
trifft voll zu	trifft eher zu	teils-teils	trifft weniger zu	trifft nicht zu	kann ich nicht beurteilen

---

6.) Ich würde eine Video-Konferenzsoftware mit solcher Bildqualität weiterverwenden:

☐	☐	☐	☐	☐	☐
trifft voll zu	trifft eher zu	teils-teils	trifft weniger zu	trifft nicht zu	kann ich nicht beurteilen

---

7.) Unabhängig von Ihrer Antwort auf die vorhergehende Frage: Ich würde mich nach alternativer Video-Konferenzsoftware umsehen:

☐	☐	☐	☐	☐	☐
trifft voll zu	trifft eher zu	teils-teils	trifft weniger zu	trifft nicht zu	kann ich nicht beurteilen

---

8.) Ich wäre bereit, die Qualitätseinbußen für einzelne Sessions zu tolerieren, wenn ich wüsste, dass dadurch die Qualität für zukünftige Video-Konferenzen verbessert wird:

☐	☐	☐	☐	☐	☐
trifft voll zu	trifft eher zu	teils-teils	trifft weniger zu	trifft nicht zu	kann ich nicht beurteilen

---

In Video-Konferenzsystemen kommt es häufiger mal vor, dass sich durch die verlustbehaftete Kompression Artefakte bilden. Diese zeichnen sich durch das Entstehen von Block-Artefakten und einer gewissen Unschärfe aus.

9.) Die hier gezeigten Artefakte sind störender als typische Kodierungsartefakte:

☐	☐	☐	☐	☐	☐
trifft voll zu	trifft eher zu	teils-teils	trifft weniger zu	trifft nicht zu	kann ich nicht beurteilen

Abbildung A.4: Ausschnitt aus dem mit dem Wasserzeichen versehenem Video: vidyo4

A.4 DIE ANTWORTEN DER PROBANDEN AUF DEN FRAGEBOGEN

Nr. des Probanden	1. Frage	2. Frage	3. Frage	4. Frage	5. Frage	6. Frage	7. Frage	8. Frage	9. Frage
1	3	4	4	1	4	3	4	3	2
2	1	1	1	0	5	1	5	3	2
3	1	2	1	2	5	1	5	1	5
4	1	4	1	2	5	1	5	2	3
5	1	4	1	3	5	2	4	5	0
6	1	3	1	2	5	1	5	2	5
7	1	4	1	5	5	2	5	3	0

Abbildung A.5: Die Antwort der Probanden auf die unterschiedlichen Fragen

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LITERATURVERZEICHNIS

[1] Computerwoche „Zu 100 Prozent fehlerfreie Software gibt es nicht, “ https://www.computerwoche.de/a/zu-100-prozent-fehlerfreie-software-gibt-es-nicht,567618 , 2005 [Jan. 17, 2018]

[2] D. Schönfeld, H. Klimant, R. Piotraschke, „Informations- und Kodierungstheorie, “ Vieweg+Taubner Verlag, 2012.

[3] Iain E. Richardson, „The H.264 Advanced Video Compression Standard, “ Wiley, 2010.

[4] Tilo Strutz. „Bilddatenkompression, “ Wiesbaden, Deutschland, Vieweg+Taubner Verlag, 2009.

[5] Rafael C. Gonzalez, Richard E. Woods, „Digital Image Processing, “ Pearson, 2007.

[6] Kent Beck, „Test - Driven Development By Example, “ Addison-Wesley Professional, 2002

[7] Peter Wayner, „Disappearing Cryptography: Information Hiding: Steganography & Watermarking “ Baltimore, MD, Elsevier Inc., 2008

[8] H. Al-Khalifa,„Utilizing QR Code and Mobile Phones for Blinds and Visually Impaired People, “ Deutschland, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, p.1065-1069

[9] G. Oberholzer, „How Long Is Too Long? The Influence of Absolute Delay on User Perception in Video Conferencing,“. Diplomarbeit, Eidgenössische Technische Hochschule Zürich, Schweiz, 2002

[10] K. A. Saadi, A. Bouridane und A. Guessoum, „Combined fragile watermark and digital signature for H.264/AVC video authentication,“ 2009 17th European Signal Processing Conference, Glasgow, 2009, pp. 1799-1803.

[11] T. Y. Kuo, Y. C. Lo und C. I. Lin, „Fragile Video Watermarking Technique by Motion Field Embedding with Rate-Distortion Minimization,“ 2008 International Conference on Intelligent Information Hiding and Multimedia Signal Processing, Harbin, 2008, pp. 853-856.

[12] G. C. Langelaar und R. L. Lagendijk, „Optimal differential energy watermarking of DCT encoded images and video,“ in IEEE Transactions on Image Processing, vol. 10, no. 1, pp. 148-158, Jan 2001.

[13] D. L. Robie and R. M. Mersereau, „Video error correction using steganography,“ in EURASIP Journal on Advances in Signal Processing, 2002, pp. 164 –173

[14] S. K. Kapotas, E. E. Varsaki und A. N. Skodras, „Data Hiding in H. 264 Encoded Video Sequences,“ 2007 IEEE 9th Workshop on Multimedia Signal Processing, Crete, 2007, pp. 373-376.

[15] D. Pröfrock, M. Schlauweg, und E. Müller, „A New Uncompressed Domain Video Watermarking Approach Robust to H.264/AVC Compression,“ Proceeding Signal Processing, Pattern Recognition, and Applications, Innsbruck, Österreich, 2006.

[16] J. Fridrich, „Image watermarking for tamper detection,“Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269), Chicago, IL, 1998, pp. 404-408 vol.2.

[17] S. Thiemert, T. Vogel, J. Dittmann und M. Steinebach, „A high-capacity block based video watermark,“ Proceedings. 30th Euromicro Conference, 2004., 2004, pp. 457-460.

[18] E. Lin, C. Podilchuk, und E. Delp, „Detection of image alterations using semi-fragile watermarks,“ in Proc. SPIE, Security and Watermarking of Multimedia Contents II, San Jose, CA, 2000, vol. 3971, pp. 152-163.

[19] Yong-Gang Fu und Hui-Rong Wang, „Self-reference based semi-fragile image watermarking scheme for authentication,“ 2008 2nd International Conference on Anti-counterfeiting, Security and Identification, Guiyang, 2008, pp. 93-96.

[20] Minghua Chen, Yun He und R. L. Lagendijk, „A fragile watermark error detection scheme for wireless video communications," in IEEE Transactions on Multimedia, vol. 7, no. 2, pp. 201-211, 2005.

[21] S. Süsstrunk and S. Winkler, „Color image quality on the Internet," in Proc. SPIE Internet Imaging, vol. 5304, San Jose, CA, pp. 118–131, January 19–22

[22] L. Superiori, O. Nemethova, and M. Rupp, „Detection of visual impair- ments in the pixel domain of corrupted H.264/AVC packets," IEEE Int. Picture Coding Symposium, Lisbon, Nov. 2007.

[23] G. Sullivan und A. Artino Jr „Analyzing and Interpreting Data From Likert-Type Scales," Journal of Graduate Medical Education, Vol. 5, No. 4, pp. 541-542, December 2013.

[24] Cyril Kowaliski, „New Intel IGP drivers add H.265, VP9 hardware decode support," http://techreport.com/news/27677/new-intel-igp-drivers-add-h-265-vp9-hardware-decode-support, Jan. 15, 2015 [Jan. 10, 2018]

[25] Global Media Format Report 2017, http://1yy04i3k9fyt3vqjsf2mv610yvm-wpengine.netdna-ssl.com/files/2017-Global-Media-Formats-Report.pdf, [Okt. 30, 2017]

[26] „Xiph.org Video Test Media [derf's collection] ", Internet: https://media.xiph.org/video/derf/ [Jan. 17, 2018]

[27] „QR Code Essentials", Internet: http://www.nacs.org/LinkClick.aspx?fileticket=D1FpVAvvJuo%3D&tabid=1426&mid=4802, 2011 [Dez. 12, 2017]

[28] „NumPy " Internet: http://www.numpy.org/, 2017 [Jan. 17, 2018]

[29] „scipy.fftpack.dct ", Internet:https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.fftpack.dct.html, [Dez. 17, 2017]

[30] „scipy.fftpack.idct ", Internet: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.fftpack.idct.html, [Dez. 17, 2017]

[31] David Zax, „Many Cars Have a Hundred Million Lines of Code ", Internet: https://www.technologyreview.com/s/508231/many-cars-have-a-hundred-million-lines-of-code/ Dez 3, 2012, [Jan. 17, 2018]

[32] Ryan Paul, „Linux kernel in 2011: 15 million total lines of code and Microsoft is a top contributor ", Internet: https://arstechnica.com/information-technology/2012/04/linux-kernel-in-2011-15-million-total-lines-of-code-and-microsoft-is-a-top-contributor/, 04.04.2012 [Dez. 18, 2017]

[33] Jeff Desjardins, „Here's how many millions of lines of code it takes to run different software ", Internet: http://www.businessinsider.de/how-many-lines-of-code-it-takes-to-run-different-software-2017-2?r=US&IR=T, 10.02.2017 [Jan. 17, 2018]

[34] „H.264 Video Encoding Guide ", Internet: https://trac.ffmpeg.org/wiki/Encode/H.264, November 1, 2007 [Jan. 17, 2018]

[35] „matplotlib.axes.Axes.boxplot ", Internet: https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.boxplot.html [Dez. 19, 2017]

[36] „Time access and conversions ", Internet: https://docs.python.org/2/library/time.html#time.time [Dez. 19, 2017]