Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jannik Schick

ROI basierte selektive Kompression für JPEG Anwendungen im Automobilbereich

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jannik Schick

ROI basierte selektive Kompression für JPEG Anwendungen im Automobilbereich

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Korf
Zweitgutachter: Prof. Dr.-Ing. Meisel

Eingereicht am: 11. Februar 2016

Jannik Schick

Thema der Arbeit
ROI basierte selektive Kompression für JPEG Anwendungen im Automobilbereich

Stichworte
JPEG, Bildkompression, Region of Interest, selektive Kompression, Echtzeit Ethernet

Kurzzusammenfassung
Diese Bachelorarbeit befasst sich mit Region of Interest (ROI) basierter selektiver Bildkompression, aufbauend auf dem JPEG Bildkompressionsstandard. Es werden zwei Konzepte entwickelt, eine ROI basierte Kompression mit Hilfe des JPEG Standards umzusetzen. Diese Kompressionen werden in einem Versuchsfahrzeug, das mit Kameras, Computern und einem Echtzeit Ethernet ausgestattet ist, zur Anwendung gebracht. Die zwei entwickelten Konzepte werden dabei in einem Programm umgesetzt, das neben der Kompression außerdem die zu komprimierenden Bilder aufnimmt und über Echtzeit Ethernet verschickt. Am Ende werden die beiden Konzepte miteinander verglichen und bewertet.
Jannik Schick

Title of the paper
ROI based selective compression for JPEG applications in the automotive sector

Keywords
JPEG, image compression, Region of Interest, selective compression, real-time ethernet

Abstract
This bachelor thesis deals with region of interest (ROI) based selective image compression, based on the JPEG image compression standard. Two concepts to implement an ROI based compression with the JPEG standard, will be developed. These compressions will be applied in a test vehicle, equipped with cameras, computers and a real-time ethernet. The two concepts developed, are going to be implemented in a program that besides compressing, also captures the images and sends them over real-time ethernet. Finally the two concepts will be evaluated and compared to each other.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

In modernen Automobilen gibt es eine immer größer werdende Anzahl von Computern und computergesteuerten Fahrerassistenzsystemen. Viele dieser Assistenzsysteme arbeiten mit einer oder mehreren Kameras, die um das Fahrzeug herum angebracht sind. Da die durch Kameras erzeugte Datenmenge sehr groß ist, sind auch die Anforderungen an die Hardware, die diese Daten verarbeitet, recht hoch. Zusätzlich zur erhöhten Rechenkapazität, wird auch eine hohe Bandbreite benötigt, um die Daten im Fahrzeugnetzwerk zu verschicken. Zur Reduktion der Datenmenge, kann neben einer normalen Bildkompression auch noch eine sogenannte Region of Interest(ROI) eingesetzt werden. Mit einer ROI können Ausschnitte eines Bildes markiert werden, die für die Weiterverarbeitung wichtig sind. Bei der Kompression mit einer ROI können dann die Bildbereiche außerhalb der ROI stärker komprimiert werden. Ziel der Arbeit ist es, ein Konzept für eine solche ROI basierte selektive Komprimierung zu entwickeln und umzusetzen. Die Umsetzung findet mit Hilfe eines Versuchsfahrzeugs statt, welches unter anderem von der CoRE (Communication over Real-Time Ethernet) Projektgruppe (siehe CoRE Research Group) der HAW Hamburg entworfen wurde. In diesem Fahrzeug sind Kameras sowie Rechner zur Kompression der Bilder verbaut. Neben dem Ziel der Datenreduktion gibt es noch weitere Rahmenbedingungen, die zu berücksichtigen sind. Zum einen hat die für die Kompression zur Verfügung stehende Hardware begrenzte Rechenkapazität, zum anderen müssen Anforderungen, die durch das im Fahrzeug verwendete Real-Time Ethernet gestellt werden, erfüllt werden.

Gliederung der Arbeit

In Kapitel 2 werden Grundlagen dieser Arbeit erläutert. Es wird eine Einführung in die Bildkompression gegeben und genauer auf das JPEG Kompressionsverfahren eingegangen. Des Weiteren werden die in dieser Arbeit verwendeten Werkzeuge vorgestellt und die Grundlagen von Echtzeitnetzwerken erläutert. In Kapitel 3 wird analysiert, welche Anforderungen und Ziele diese Arbeit hat. Auf dieser Basis werden verschiedene Bildkompressionsverfahren verglichen und die bisher bestehende Implementierung betrachtet. In Kapitel 4 werden unterschiedliche bestehende Konzepte zur Umsetzung von ROIs betrachtet und zwei Konzepte entwickelt, die in

dieser Arbeit umgesetzt werden. Kapitel 5 befasst sich mit der Implementierung der Konzepte und des gesamten Programms. Wie das Programm getestet wird, wird in Kapitel 6 beschrieben. In Kapitel 7 werden die umgesetzten Konzepte verglichen und die Ergebnisse der Arbeit bewertet. Zum Abschluss wird in Kapitel 8 ein Fazit gezogen.

2 Grundlagen

In diesem Kapitel werden die Grundlagen, die zum Verständnis dieser Arbeit beitragen, erläutert. Es werden Bildkomprimierung, der Unterschied zwischen Interframe und Intraframe Kompressionsverfahren, sowie das JPEG Kompressionsverfahren erklärt. Des Weiteren wird kurz auf die in dieser Thesis verwendete Hardware sowie Arbeitsmittel eingegangen und es werden die Eigenschaften von Echtzeitnetzwerken erläutert.

2.1 Bildkomprimierung

Bei der Verarbeitung von Bildern können, wenn sie im Rohformat gespeichert werden schnell riesige Datenmengen anfallen. Ein Bild mit einer Auflösung von 1000x1000 Pixeln und 8 Bit Farbtiefe pro Farbkanal, hat eine Dateigröße von 3MB(siehe Formel 2.1), wenn drei Farbkanäle(z.B. RGB) verwendet werden.

$$AufloesungX * AufloesungY * nFarbkanaele * nBitFarbtiefe = Dateigroesse \quad (2.1)$$

Wird die Auflösung auf 2000x2000 Pixel verdoppelt, vervierfacht sich die Datenmenge. Problematisch wird dies im Bereich von Videos, wo oft mit 24 oder auch mehr Bildern pro Sekunde gearbeitet wird. Bei solchen Datenmengen werden schnell die Speicherkapazitäten von Speichermedien oder die verfügbaren Bandbreiten zur Übertragung überschritten, deshalb ist bei den meisten Anwendungsfällen eine Komprimierung der Daten zwingend notwendig. Grundsätzlich wird zwischen zwei Komprimierungen unterschieden, verlustbehafteten und verlustfreien. Bei verlustfreien Komprimierungsverfahren, sind die Originaldaten und die nach der Komprimierung wieder dekomprimierten Daten identisch. Verlustbehaftete Komprimierungsverfahren erzeugen Bilder, die geringe Abweichungen beispielsweise in den Farbwerten haben. Das Ziel von verlustbehafteten Verfahren ist es, die Datenmenge ohne visuelle Verschlechterung der Bildqualität so stark wie möglich zu verringern.

2.1.1 Inter- und Intra-frame Komprimierungsverfahren

Bei der Videokomprimierung wird zusätzlich zur Unterscheidung zwischen verlustbehafteten und verlustfreien auch noch zwischen Interframe und Intraframe Verfahren unterschieden. Im folgenden werden diese Unterschiede erläutert.

In den sogenannten Intraframe Komprimierungsverfahren(Motion-JPEG(siehe ISO (1994)), Motion-JPEG2000(siehe ISO (2000)), u.a.) wird jeder Frame einzeln mit dem entsprechendem Komprimierungsalgorithmus komprimiert, ohne dass Informationen anderer Bilder in die Komprimierung mit einfließen. Dies hat z.B. den Vorteil, dass die Komprimierung weniger rechenaufwändig und damit schneller ist, allerdings wird mögliches Kompressionspotenzial, das durch mögliche Gemeinsamkeiten zwischen Frames entsteht, vergeben.

Interframe Komprimierung(MPEG-1(siehe ISO (1993)), H.264(siehe ISO (2003)), u.a.) kann neben den Bildinformationen des aktuellen Frames, auch Bildinformationen vorangegangener Frames und kommender Frames (sofern diese schon vorliegen) berücksichtigen. So kann eine wesentlich höhere Kompressionsrate als bei Intraframe-Verfahren erreicht werden. Unterschieden wird zwischen I(intra)-Frames, P(predicted)-Frames und B(bidirectional)-Frames, die die folgenden Eigenschaften haben(siehe Salomon und Motta (2010)):

I-Frames werden komplett einzeln, ohne einbeziehen von Daten anderer Frames komprimiert.

P-Frames beziehen Daten anderer vorangegangener Frames mit ein. Z.B. kann bei einem Bildbereich, der sich zum vorangegangenen Frame nicht verändert hat, auf diesen verwiesen werden, und der Bereich muss nicht mehr komprimiert und übertragen werden.

B-Frames enthalten wie P-Frames Verweise auf andere Frames, haben jedoch nicht nur Verweise auf einen vorangegangenen Frame, sondern auch auf einen zukünftigen. Diese Eigenschaft macht sie für zeitkritische Realtime-Anwendungen, bei denen Frames so schnell wie möglich verschickt werden müssen, allerdings unbrauchbar.

Die unterschiedlichen Frame Typen haben zur Folge, dass der Kompressionsaufwand deutlich erhöht wird, weil bei der Komprimierung eines Frames zusätzlich die Daten eines oder sogar mehrerer anderer Frames vom Algorithmus verarbeitet werden müssen. Des Weiteren haben die Frame Typen Auswirkungen auf die Netzwerkauslastung, die bei Intraframe Verfahren durch die immer in etwa gleich großen Frames relativ gleichmäßig ist. Bei Interframe Verfahren ist die Netzwerkauslastung unregelmäßig und nicht gut vorhersagbar, I-Frames erzeugen Spitzen in der Netzwerkauslastung und P/B-Frames können je nach Stärke der Veränderung des Bildinhalts in der Größe variieren. Die Größe von Frames kann zwar auch bei Intraframeverfahren je nach Bildinhalt schwanken, allerdings nicht so stark wie bei Interframeverfahren.

2.1.2 JPEG/Motion-JPEG

Die Komprimierung von RGB Bildern mithilfe des JPEG(Joint Photographer Experts Group)-Algorithmus ist ein weit verbreitetes, verlustbehaftetes Kompressionsverfahren zur Komprimierung von Einzelbildern. Unter dem Namen Motion-JPEG wird es auch verwendet, um Videos zu komprimieren, der Algorithmus verändert sich hierbei nicht. Es werden pro Bild die folgenden fünf Schritte auf jeden einzelnen Videoframe angewendet. (siehe Salomon und Motta (2010) und Tanenbaum (2009))

Umrechnung des Farbraums Im ersten Schritt werden die Rohdaten, die meist im RGB Format vorliegen in das YC_bC_r Format umgewandelt. In dieser Darstellung gibt Y die Helligkeit, C_b die „Color blueness“ und C_r die „Color redness“ an. In diesem Schritt findet keine Kompression statt, es kann jedoch durch Rundung zu Informationsverlust/-veränderung kommen.

Farbunterabtastung In diesem Schritt wird die Eigenschaft des menschlichen Auges, Helligkeitsunterschiede stärker wahrzunehmen als Farbunterschiede, ausgenutzt. Durch die Umwandlung in das YC_bC_r Format können in den beiden Farbanteilen jeweils von benachbarten Werten der Mittelwert gebildet, und so die Auflösung dieser Anteile verringert werden. Je nach gewählter Unterabtastung, trägt dieser Schritt stärker oder weniger stark zur Kompression bei. Eine übliche Variante der Farbunterabtastung ist, vier Pixelwerte wie in Abbildung 2.1 zusammenzufassen. In diesem Fall wird die Datenmenge in den Farbanteilen um 75% reduziert.

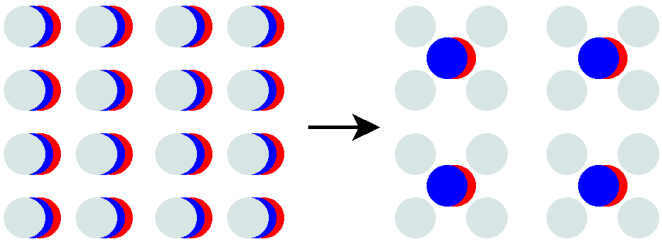

Abbildung 2.1: Beispiel für die Farbunterabtastung.

Diskrete Kosinus Transformation Um diesen Schritt durchzuführen, wird zuerst das Bild in 8x8 Pixelblöcke eingeteilt. Auf diese Blöcke wird für jede Bildkomponente die Diskrete Kosinus Transformation(DCT) angewandt. Dabei werden nicht die Informationen an sich, aber die Darstellung dieser verändert. Es werden nicht mehr die Werte der einzelnen Pixel gespeichert, sondern das Frequenzspektrum des Blocks. Die Werte der daraus resultierenden 8x8 Matrix stellen die Helligkeitswechsel des Blocks dar, im horizontalen sowie

im vertikalen, und werden DCT-Koeffizienten genannt. Die langsamsten Helligkeitswechsel (tiefe Frequenzen) stehen in der Matrix oben links. Je weiter in der Matrix nach rechts bzw. nach unten gegangen wird, desto schneller werden die Helligkeitswechsel (hohe Frequenzen). In Abbildung 2.2 ist Verteilung der Helligkeitswechsel dargestellt. Dieser Schritt wird für jede Bildkomponente durchgeführt.

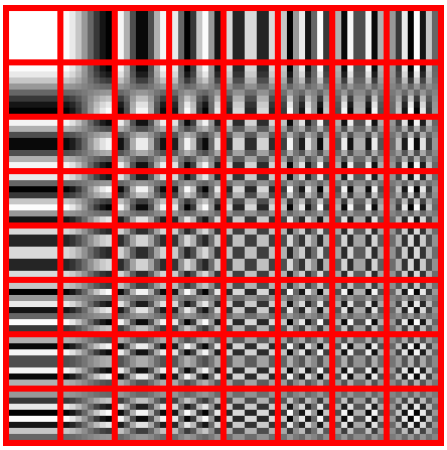

Abbildung 2.2: Beispielhafte Darstellung der Frequenzverteilung in einem 8x8 Pixelblock. Quelle: Wikimedia

Quantisierung Als nächstes werden die Blöcke mit einer vorher festgelegten 8x8 Quantisierungsmatrix quantisiert. Das bedeutet, dass die DCT-Koeffizienten eines Blocks durch den entsprechenden Wert der Quantisierungsmatrix geteilt werden(siehe Abbildung 2.3). Da Ganzzahlen gespeichert werden, werden die Nachkommastellen ignoriert. Dabei werden tiefe Frequenzen durch kleinere Werte geteilt als hohe Frequenzen, um die Werte der hohen Frequenzen stärker zu verringern, und hier mehr Nullen entstehen zu lassen. Dies geschieht, um hohen Frequenzen weniger Relevanz zuzuordnen. Die Priorisierung der tiefen Frequenzen beruht auch, wie bei der Farbunterabtastung, auf den speziellen Eigenschaften des menschlichen Auges. Die Qualität des komprimierten Bildes und die Stärke der Kompression hängen hauptsächlich von der gewählten Quantisierungsmatrix ab. Je höher die Werte in der Quantisierungsmatrix, desto mehr Nullen entstehen beim Quantisieren. Die Anzahl der Nullen wirkt sich maßgeblich auf die Kompressionsrate, die durch den Codierungsschritt erreicht werden kann, aus. Auch in diesem Schritt findet ein Informationsverlust statt: Je höher die Werte der Quantisierungsmatrix, desto stärker die Komprimierung und der Qualitätsverlust.

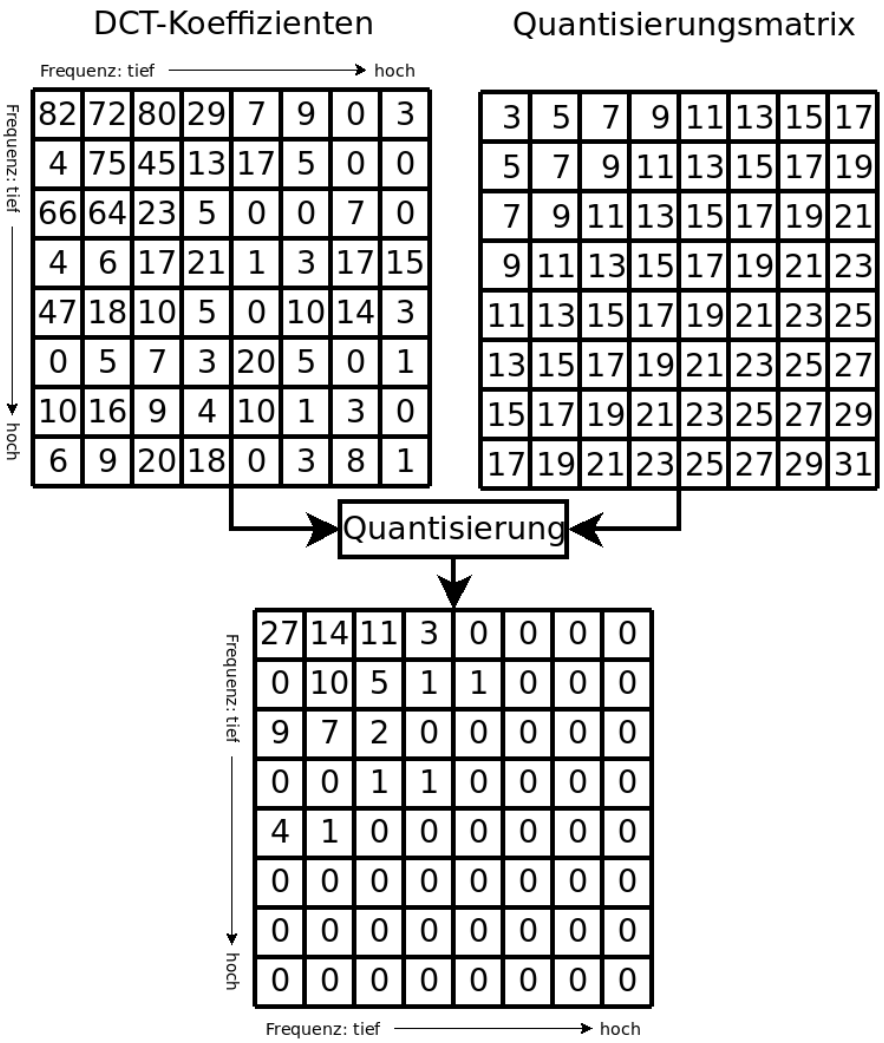

Abbildung 2.3: Beispiel für eine Quantisierungsmatrix, DCT-Koeffizienten und die daraus resultierende quantisierte Matrix.

Codierung Für die Codierung werden die Werte eines Blocks erst in eine andere, geeignetere Reihenfolge umsortiert. Die Werte werden von [0,0] bis [7,7] in einem „Zickzackkurs“ abgetastet([0,0;1,0;0,1;0,2;1,1;2,0; ...]), siehe Abbildung 2.4. Diese Sortierung hat zur Folge, dass die Nullen, die bei der Quantisierung entstehen, gehäuft am Ende des Arrays auftreten. Auf die sortierten Werte werden dann eine Lauflängenkodierung und Huffman-

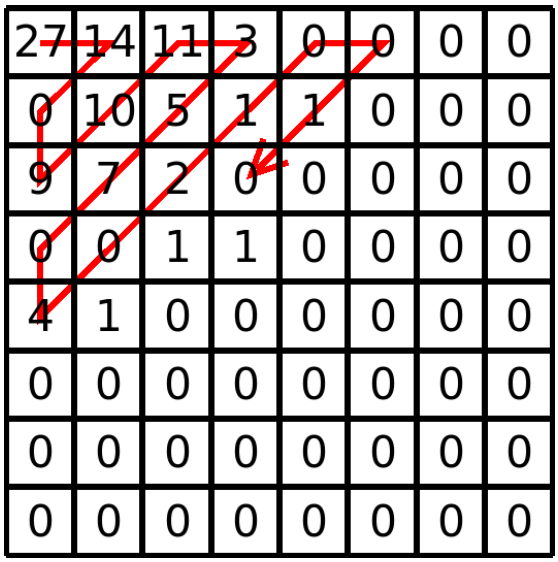

Abbildung 2.4: Reihenfolge, in der die Werte für die Codierung abgetastet werden.

Kodierung angewandt, die die vorhandenen Datenmenge erheblich reduzieren(siehe Wallace (1991)). Die Lauflängenkodierung im JPEG Algorithmus speichert nicht jeden Wert einzeln, sondern nur jeden Wert, der nicht Null ist und dazu die Anzahl der diesem Wert vorangegangenen Nullen. Besteht das Ende eines Blocks nur aus Nullen, werden diese durch ein End-Of-File Symbol ersetzt. Die Huffman-Kodierung sucht in den Daten nach häufig vorkommenden Werten und bildet diese auf weniger Bit ab, als seltener vorkommende Werte. So kann die Datenmenge zusätzlich zu der Reduzierung durch die Lauflängenkodierung, nochmals reduziert werden.

2.1.3 Region of Interest Komprimierung

Als Region of Interest wird der Bereich eines Bildes genannt, der für den Betrachter oder für die Weiterverarbeitung von besonderem Interesse ist. Aufgrund dessen, wird dieser Bereich im Bild anders komprimiert, als der Rest des Bildes, der oft als Hintergrund bezeichnet wird. Der Unterschied besteht in den meisten Fällen darin, dass die Region of Interest (ROI) weniger

starker Kompression unterzogen wird, als der Hintergrund. Möglich ist auch, dass die ROI gar nicht oder mit einem anderen Kompressionsverfahren komprimiert wird. Einige bereits vorhandene und die für diese Arbeit verwendeten Verfahren, werden im Kapitel 4 Konzept genauer erläutert.

2.2 Hardware

Diese Arbeit wurde im Rahmen des RECBAR (Realtime Ethernet Backbone for Cars) Projekts angefertigt. Das Projekt besteht aus einem VW Golf, in den Sensoren und Computer eingebaut sind, die über ein Echtzeit Ethernet vernetzt sind. In diesem Abschnitt werden die einzelnen Hardwarekomponenten, die in dieser Arbeit verwendet werden vorgestellt. Außerdem wird der Aufbau der Komponenten und des Netzwerks im RECBAR Versuchsfahrzeug erklärt.

2.2.1 Kameras

Zur Aufnahme der Bilder, kommen zwei Kameras vom Typ uEye UI-1240ML-C, des Herstellers IDS Imaging Development Systems GmbH zum Einsatz. Sie verfügen über einen CMOS Farbsensor, mit einer Auflösung von 1280x1024 Pixeln. Die maximal verfügbare Framerate beträgt 25 fps. Die Kommunikation mit den Kameras findet über USB statt, auch die Stromversorgung wird über USB realisiert. Über eine API können Funktionen und Parameter softwareseitig gesteuert werden, die API wird in Abschnitt 2.3.1 der Grundlagen näher erläutert. (siehe IDS Imaging (b))

2.2.2 Hardware für Kameragateways

Die Verarbeitung und Komprimierung der von der Kamera gelieferten Rohdaten findet auf Boards vom Typ pITX-SP 2.5“ SPC, des Herstellers Kontron statt. Die Boards haben eine Intel Atom CPU vom Typ Z510 bzw. Z530 und sind mit 1,1 bzw. 1,6 GHz getaktet. Es handelt sich um Single-Core Prozessoren. Die Boards verfügen über USB Anschlüsse zum Anschluss der Kamera und einen Gigabit Ethernet Anschluss zum Versenden des komprimierten Bilds über das Netzwerk. Im folgenden werden diese Boards als Kameragateways bezeichnet. (siehe Kontron AG)

2.2.3 Aufbau des Systems

Im RECBAR Versuchsfahrzeug sind die Kameras in der Front und im Heck des Fahrzeugs angebracht und per USB an das dazugehörige Kameragateway angeschlossen. In Abbildung

2.5 ist schematisch der Aufbau des Fahrzeugnetzes dargestellt. Das Gateway mit der größeren

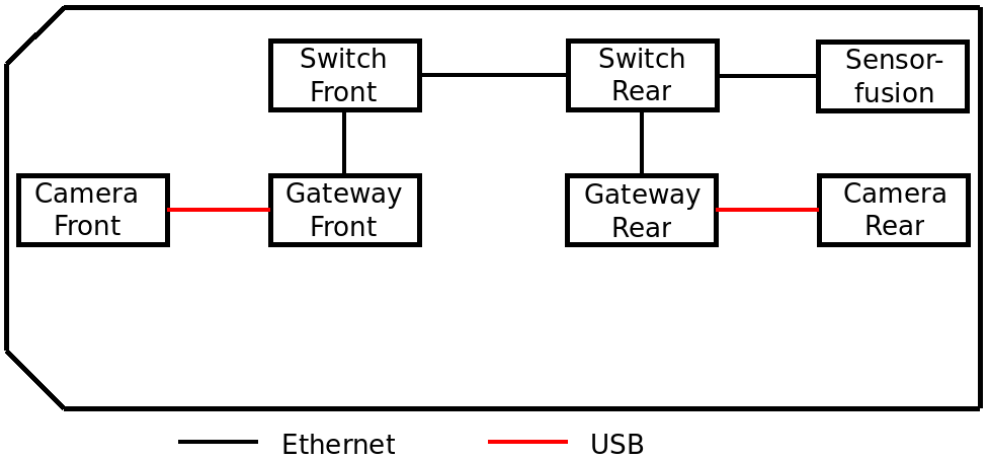

Abbildung 2.5: Schematischer Aufbau des Fahrzeugnetzes, mit den für diese Arbeit relevanten Komponenten.

Rechenleistung befindet sich in der Front des Fahrzeugs. Die Gateways sind mit 100 Mbit/s Ethernet an den Echtzeitethernetswitch des entsprechenden Fahrzeugteils angeschlossen. Im hinteren Teil des Fahrzeugnetzes befindet sich außerdem ein Sensorfusionsrechner, der die komprimierten Bilder der Kameragateways empfängt.

2.3 Arbeitsmittel

In diesem Abschnitt werden einige Arbeitsmittel, die für die Umsetzung dieser Arbeit benutzt wurden vorgestellt.

2.3.1 uEye Kamera-API

Die uEye Kamera API bietet Anwendungsentwicklern die Möglichkeit, aus dem Programm heraus auf die Kamera zuzugreifen, Einstellungen vorzunehmen und aufgenommene Bilder abzurufen. Über die API lassen sich Betriebs- und Aufnahmemodi einstellen. Außerdem lassen sich Framerate, Belichtungszeit, Auflösung und weitere Aufnahmeparameter einstellen. Des Weiteren können Informationen über den Status der Kamera abgefragt und Fehlermeldungen abgerufen werden. (siehe IDS Imaging (a))

2.3.2 libjpeg-turbo

Libjpeg ist eine Bibliothek zum Codieren und Decodieren von JPEG Bildern. Sie wurde initial von Tom Lane und der Independent JPEG Group(IJG) entwickelt. Libjpeg-turbo baut auf der originalen Libjpeg Bibliothek auf und bietet durch einige Verbesserungen, Geschwindigkeitsvorteile gegenüber der Standard Libjpeg. Die Bibliothek bietet neben der Möglichkeit Komprimierungseinstellungen wie Qualität, Farbunterabtastung und DCT Algorithmus festzulegen, auch weiterführende Einstellungen. Es ist beispielsweise möglich, eigene Quantisierungsmatrixen festzulegen, oder den JPEG Header zu bearbeiten. (siehe libjpeg-turbo (b))

2.3.3 Qt Framework

Das Qt Framework ist ein quelloffenes C++ Framework zur Entwicklung von GUI und Kommandozeilen Anwendungen. Es erleichtert die Entwicklung durch Erweiterung der standard C++ Funktionalität. Außerdem hat es den Vorteil, dass es plattformunabhängig ist und die mit diesem Framework geschriebenen Anwendungen auf vielen gängigen Betriebssystemen ausführbar sind. (siehe The Qt Company)

2.4 Echtzeitnetzwerke

Der Unterschied zwischen normalen Systemen und Echtzeitsystemen besteht darin, dass Aufgaben in einem Echtzeitsystem zu einem bestimmten Zeitpunkt abgeschlossen sein müssen. Dieser Zeitpunkt wird als Deadline bezeichnet. Kann diese Deadline nicht eingehalten werden, kann es zu Störungen, Fehlverhalten im System oder zu einem kompletten Versagen des Systems kommen. Man unterscheidet außerdem zwischen harten und weichen Echtzeitsystemen, die folgendermaßen definiert sind(siehe Tanenbaum (2009)).

Weiche Echtzeit Als weiche Echtzeitsysteme werden Systeme bezeichnet, bei denen im Normalbetrieb alle Deadlines eingehalten werden. Es können jedoch wenn das System unter hoher Last steht, Deadlines verletzt werden. Dies ist nicht erwünscht, es kommt jedoch nicht zu einem kompletten Versagen des Systems. Ein Beispiel für ein weiches Echtzeitsystem ist zum Beispiel ein Audio/Video Abspielgerät, bei dem Daten zum Abspielzeitpunkt vorliegen müssen. Kommt es zu der Verletzung einer Deadline, entstehen Verzögerungen oder Fehler beim Abspielen, es besteht jedoch keine Gefahr für das System oder dessen Umgebung.

Harte Echtzeit Als harte Echtzeitsysteme werden Systeme bezeichnet, bei denen es zu jeder Zeit des Betriebs, auch in Lastsituationen erforderlich ist, dass alle Deadlines eingehalten

werden. Würde eine Aufgabe bis zu ihrer Deadline nicht abgearbeitet werden, wäre dies gleichbedeutend damit, dass sie nie abgeschlossen wird. Wird eine harte Deadline nicht eingehalten, kann dies katastrophale Folgen für das System haben. Als Beispiel für ein hartes Echtzeitsystem, können Steuersysteme im Automobilbereich genannt werden. Können die Informationen der Sensoren über den Zustand des Fahrzeugs nicht rechtzeitig verarbeitet werden, könnte dies zu einem Unfall führen.

Wendet man diese Definition auf Netzwerke an bedeutet es, dass Ethernetframes zu einem bestimmten Zeitpunkt am Ziel eingetroffen sein müssen. Der Ethernet Standard IEEE 802.3 unterstützt keine Echtzeitanforderungen. Hierfür sind Erweiterungen notwendig, wie z.B. Time-Triggered Ethernet, welches im folgenden Abschnitt beschrieben wird.

2.4.1 Time-Triggered Ethernet

Time-Triggered Ethernet oder TTEthernet wurde in Zusammenarbeit von TTTech, Honeywell und der TU Wien entwickelt. Es erweitert das Standard Ethernet insoweit, dass neben normalem Best-Effort Verkehr auch Echtzeitverkehr möglich ist. TTEthernet definiert drei verschiedene Nachrichtentypen(siehe Steiner (2008)).

best-effort Best-Effort Traffic entspricht Standard Ethernet Traffic. Es gibt keinerlei Zusicherung, dass Pakete zu bestimmten Zeiten übertragen und zugestellt werden oder überhaupt zugestellt werden. Best-Effort Nachrichten haben im TTEthernet die geringste Priorität.

rate-constraint Rate-Constraint Traffic hat im TTEthernet eine mittlere Priorität. Es sichert einer Anwendung eine garantierte Bandbreite im Netzwerk zu.

time-triggered Time-Triggered Traffic wird zu garantierten Zeitpunkten und mit minimaler zeitlicher Verzögerung übertragen. Um dies zu erreichen werden die Übertragungszeitpunkte von Time-Triggered Nachrichten im Voraus festgelegt, und sie haben die höchste Priorität. Sie unterliegen harten Echtzeitanforderungen.

Mit Hilfe von TTEthernet können sich Netzwerke in Systemen wie Fahrzeugen und Flugzeugen, die wegen ihrer Echtzeitanforderungen nicht auf Standard Ethernet aufbauen können, mit Ethernet umsetzen lassen.

3 Analyse

Dieses Kapitel befasst sich mit der Analyse des Themas dieser Arbeit. Es werden die grundlegenden Anforderungen der Aufgabenstellung sowie die bereits bestehende Implementation erklärt und die Probleme dieser Implementation aufgezeigt.

3.1 Ziele und Anforderungen dieser Arbeit

Das grundlegende Ziel dieser Arbeit ist es ein Programm zu entwickeln, welches RAW Bilder von einer Kamera abruft, diese mit einer Region of Interest komprimiert und dann über ein Echtzeitnetzwerk versendet. Daraus ergeben sich folgende Anforderungen:

- Die durch die Anwendung erzeugte Netzwerklast, soll mithilfe der Region of Interest Komprimierung verringert werden.
- Die durch diese Anwendung erzeugte Netzwerklast soll so konstant wie möglich gehalten werden. Dies ist erforderlich, damit in Echtzeitnetzwerken nicht durch plötzliches Auftreten hoher Netzwerklast, Deadlines nicht mehr eingehalten werden können. In Netzwerken wie z.B. in Fahrzeugen, kommt es gerade in Ausnahmesituationen, zu erhöhtem Nachrichtenaufkommen. Besonders in diesen Situationen ist es wichtig, dass alle Nachrichten in der für sie vorgegebenen Zeit ankommen, und sie nicht durch ein erhöhtes Aufkommen von Videopaketen verzögert werden.
- Es soll mit möglichst wenig Rechenaufwand komprimiert werden, um auch auf der für diese Arbeit verwendeten Hardware mit geringer Rechenleistung hohe Frameraten zu erzielen.
- Das Delay von der Aufnahme bis zum versenden des Bildes, soll so gering wie möglich gehalten werden.

3.2 Auswahl des Kompressionsverfahrens

Betrachtet man verschiedene Kompressionsverfahren, unter den in Abschnitt 3.1 definierten Anforderungen, ist der Vergleich zwischen Inter- und Intraframe Kompressionsverfahren am bedeutsamsten. Interframe Verfahren verringern die Datenmenge zwar stärker als Intraframe Verfahren, jedoch gibt es große Unterschiede in der Datenmenge zwischen den verschiedenen Frametypen(siehe Abschnitt 2.1.1). In Abbildung 3.1 ist ein Vergleich der Framegrößen zwischen Motion-JPEG und MPEG4 zu sehen. Für diesen Vergleich wurde ein fünf Sekunden langes

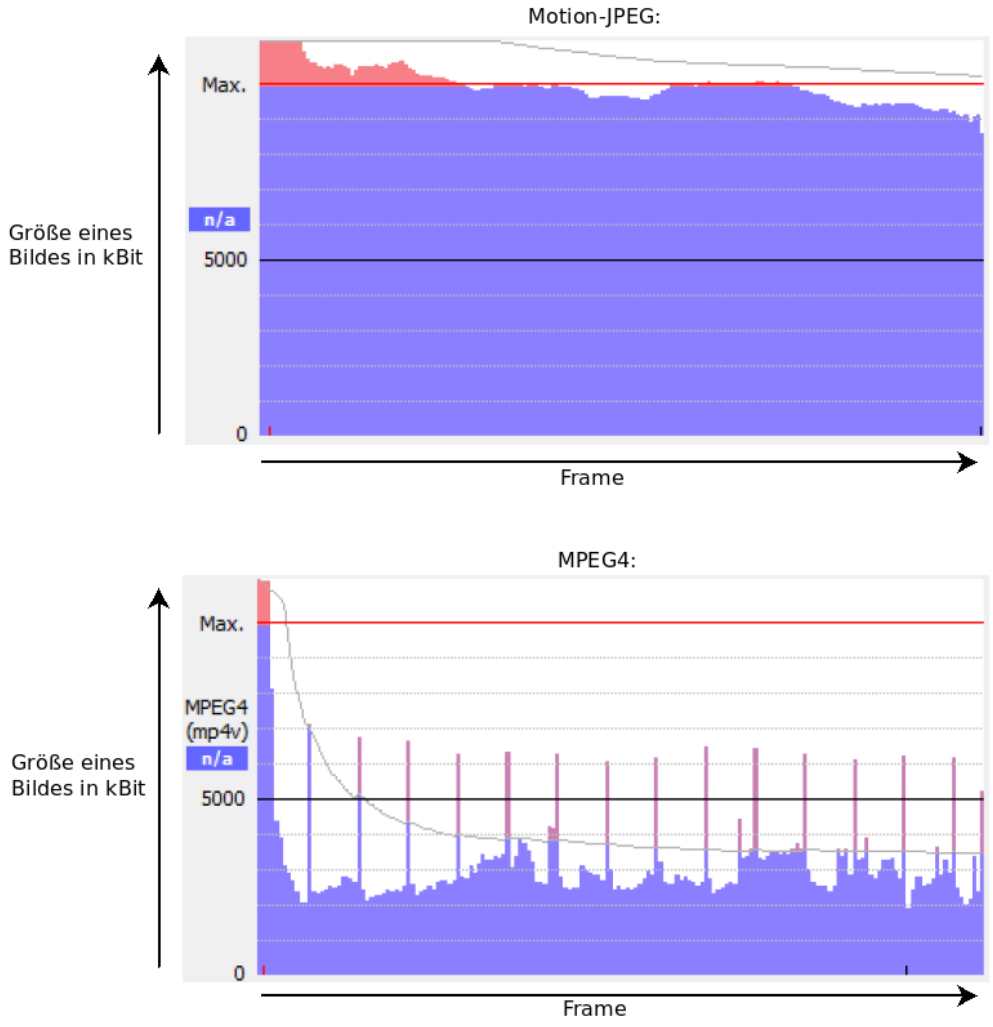

Abbildung 3.1: Vergleich der Framegrößen zwischen M-JPEG und MPEG4.

Video in die oben genannten Formate konvertiert und mit dem Programm Bitrate Viewer(siehe Konran Udo Gerber) geöffnet. In der Abbildung ist folgendes zu sehen:

- Die Y-Achse gibt die Größe eines Frames in kBit an und die X-Achse den Frame.
- Die rote horizontale Linie gibt historisch bedingt, die maximale Bitrate einer DVD von 10.000 kBit an. Die schwarze horizontale Linie entspricht der Hälfte.
- Der graue Graph gibt die durchschnittliche Größe aller bisherigen Frames an. Kommt ein Frame über diesen Graph, wird dieser Bereich des Frames farblich von dem Bereich unter dem Graph abgesetzt.
- Bei den größeren Frames im unteren Bild, die über die schwarze Linie hinaus ragen, handelt es sich um I-Frames.

Aus der Abbildung werden die Unterschiede zwischen I-Frames und P-/B-Frames des Interframeverfahrens deutlich. Ein weiteres Problem für die Anforderung einer konstanten Netzwerklast ist jedoch, dass B- und P-Frames untereinander selbst auch deutliche Größenunterschiede aufweisen können. Je stärker die Veränderung von Frame zu Frame ist, desto größer kann ein P-/B-Frame werden. Dies kann theoretisch von der Größe eines kompletten I-Frames, bei einer kompletten Bildveränderung, bis zu fast keinen Daten, bei keinerlei Veränderung reichen. Aufgrund dessen, spricht die Anforderung einer konstanten Netzwerklast, gegen ein Interframe Verfahren. Ein weiteres Argument gegen ein Interframe Verfahren ist, dass diese durch die nötigen Vergleiche zwischen unterschiedlichen Frames rechenaufwändiger sind als Intraframe Verfahren. Im Paper von Golston (2004) wurden verschiedene Kompressionsverfahren verglichen. Unter anderem wurden JPEG, MPEG4(Simple profile) und MPEG2(Main Profile) in ihrer Performance verglichen. Dabei stellte man fest, dass die Kompression mit MPEG4 mehr als doppelt und MPEG2 fast viermal so aufwändig ist wie JPEG. Aufgrund dieser Eigenschaften, wurde sich in dieser Arbeit für die Verwendung eines Intraframe Kompressionsverfahrens entschieden, im Speziellen für die JPEG Methode, weil diese am weitesten verbreitet ist.

3.3 Analyse der bisherigen Implementierung

Die bisherige Implementierung ruft Bilder von der Kamera ab, komprimiert sie als Standard JPEG und versendet diese danach. Nach dem Start des Programms wird als erstes die Kamera initialisiert. Bei der Initialisierung wird ein Auschnitt mit einer Auflösung von 960x720 Pixeln aus der Mitte des Kamerabildes ausgewählt, um den Bildbereich einzustellen, der später als Bild geliefert werden soll. Des Weiteren werden Bildspeicher angelegt, in denen Raw Bilder abgelegt

werden können. Die Kamera wird im sogenannten Bitmap-Bilddarstellungsmodus gestartet, damit die Bilder in diesen Speichern abgelegt werden. Nach erfolgreicher Initialisierung werden zwei Threads gestartet, der Erste Thread ist für das Abrufen und Komprimieren des Raw Bildes verantwortlich, der Zweite für das Versenden des JPEG über Ethernet. Bilder werden vom Aufnahmethread im sogenannten Snapshot-Bildaufnahmemodus abgerufen. In diesem Modus nimmt die Kamera mit einer konstanten Framerate Bilder auf, es wird jedoch erst ein Bild übertragen, wenn dies vom Programm angefordert wird. Es wird immer das Bild übertragen, welches als Nächstes fertig belichtet ist(siehe Abbildung 3.2). Nach dem Abrufen des Bildes

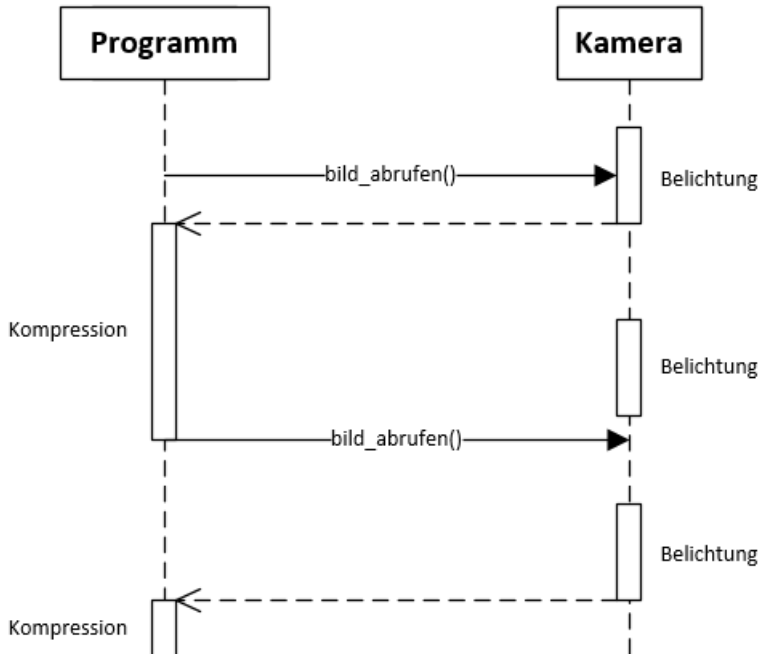

Abbildung 3.2: Sequenzdiagramm zum Verhalten des Snapshot-Bildaufnahmemodus.

wird es mithilfe der libjpeg-turbo Bibliothek komprimiert und dem Versendethread über ein Flag signalisiert, dass das Bild übertragen werden kann. Während der Übertragung wartet der Aufnahmethread. Das Versenden erfolgt über einen RAW-Ethernet-Socket, um die für das TTEthernet notwendigen Anpassungen vorzunehmen.

3.3.1 Probleme dieser Implementierung

Die bisherige Implementierung hat einige Probleme, die in Tabelle 3.1 aufgeführt werden. Problem 1 und Problem 2 hängen miteinander zusammen. Bei aktueller Auflösung entspricht die

Problem	Beschreibung
1	Gateways sind mit der Komprimierung von ca. 10fps komplett ausgelastet.
2	Bilder haben nur eine Auflösung von 960x720 Pixeln, anstatt der maximalen 1280x1024 Pixel.
3	Es besteht eine einsekündige Verzögerung zwischen Aufnahme und Anzeige des Bildes auf einem externen Gerät.

Tabelle 3.1: Auflistung der Probleme der bisherigen Implementierung.

Datenmenge eines Bildes, nur in etwa der Hälfte der Datenmenge als bei voller Auflösung. Eine Erhöhung der Auflösung würde das Problem der geringen Framrate, noch weiter verstärken.

4 Konzept

In diesem Kapitel werden einige Konzepte zur Umsetzung von Region of Interests in Bildern, bzw. variabler Komprimierung innerhalb eines Bildes vorgestellt. Da die Konzepte für sehr spezielle Anwendungsgebiete entworfen wurden, wird analysiert und geprüft, inwieweit sich Ideen aus diesen Konzepten zur Anwendung in dieser Arbeit eignen. Des Weiteren werden zwei Konzepte vorgestellt, die im Rahmen dieser Arbeit entwickelt und umgesetzt wurden. Zuletzt wird das Konzept zum Aufbau des gesammten Programms vorgestellt.

4.1 Vorhandene Konzepte zur Umsetzung einer ROI bzw. einer variablen Komprimierung

In diesem Abschnitt werden ein Konzept zur Umsetzung einer ROI und zwei Konzepte zur Umsetzung variabler Kompression vorgestellt. Diese drei Verfahren wurden wegen ihrer unterschiedlichen Ansätze ausgewählt. Das erste Konzept unterteilt das Bild abhängig vom Bildinhalt in Bereiche ein und komprimiert diese mit verschiedenen Verfahren. Das zweite Konzept unterteilt das Bild in Blöcke und untersucht deren Inhalt. Je nach Eigenschaft werden die Blöcke anders komprimiert. Das letzte Konzept komprimiert die ROI und den Hintergrund mit dem gleichen Verfahren, allerdings mit unterschiedlicher Qualität.

4.1.1 Compound Image/Video Compression

Dieser Ansatz zur variablen Komprimierung geht davon aus, dass das zu komprimierende Video unterschiedliche Inhalte, mit unterschiedlichen Anforderungen an die Kompression hat. Ein Anwendungsfall hierfür ist z.B. die Remote Desktop Übertragung, bei der der Bildschirminhalt eines Computers per Video übertragen wird. Wang u. a. (2012) unterteilen dabei in ihrem Paper das Bild in 16x16 Pixelblöcke und analysieren den Inhalt des Blocks. Unterschieden wird dabei zwischen natürlichen Bild- und Textblöcken. Danach werden die Blöcke unterschiedlichen Kompressionsverfahren zugeführt, die für den jeweiligen Inhalt gut geeignet sind. Zusätzlich wird jeder Block noch dahingehend überprüft, ob er sich zum vorangegangen Frame verändert hat und wird nur bei Veränderung übertragen.

Für die vorliegende Arbeit eignet sich dieses Verfahren nicht, da es in den Videodaten, die verarbeitet werden müssen, keine unterschiedlichen Inhalte gibt.

4.1.2 Variable Block Size Segmentation

Bei diesem Konzept zur variablen Komprimierung von unterschiedlichen Bereichen eines Bildes(siehe Vaisey und Gersho (1992)) wird das Bild anhand der lokalen Eigenschaften, wie z.B. Detailgrad, in unterschiedlich große Blöcke unterteilt. Die Aufteilung des Bildes in Blöcke wird mit Hilfe des Quadtree Algorithmus realisiert. Hierbei wird das Bild zuerst in rechteckige Blöcke von einer vorher festgelegten Größe unterteilt. Dies hat den Zweck, dass auch wenn das Bild nicht rechteckig ist, rechteckige Blöcke erzeugt werden. Im nächsten Schritt, wird jeder dieser Blöcke wieder in 4 gleich große Blöcke unterteilt. Dieser Schritt lässt sich abhängig von der Ausgangsblöckgröße mehrfach rekursiv wiederholen. In Abbildung 4.1 ist der Quadtree Algorithmus beispielhaft dargestellt.

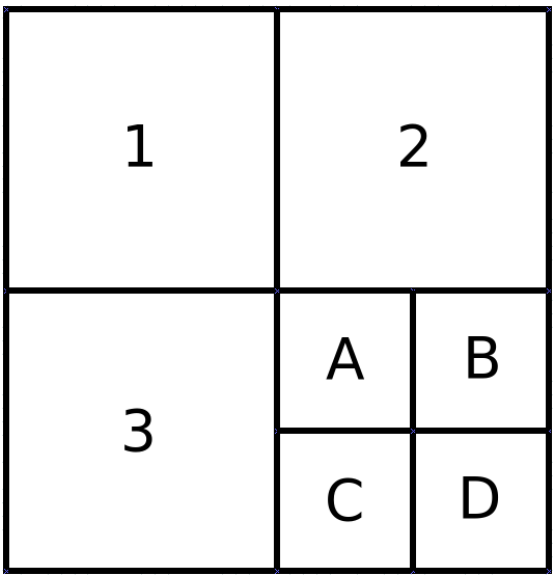

Abbildung 4.1: Beispielhafte Darstellung des Quadtree Algorithmus.

Im Paper von Vaisey und Gersho (1992) entscheidet über die Größe des Blocks der Detailgrad des Bildes in diesem Block. Sind viele Details vorhanden, wird der Block ein weiteres Mal unterteilt und die entstandenen Blöcke erneut geprüft. Um die Details zu erhalten werden die kleinsten Blöcke mit den meisten Details, mit einem anderen Verfahren komprimiert als die größeren Blöcke.

Da dieses Verfahren keine vordefinierten ROIs hat, sondern durch Bildinhalte beeinflusst wird, ist der Blockgrößenbestimmungsanteil dieses Konzepts nicht auf diese Thesis übertragbar.

4.1.3 Variable Quantization

Diese Variante zur Kompression mit einer ROI baut auf dem JPEG Algorithmus auf und greift in den Quantisierungsschritt(siehe 2.1.2) ein, um unterschiedliche Kompressionsraten zu erzeugen. Die Qualität eines mit JPEG komprimierten Bildes wird ausschlaggebend von den Werten der Quantisierungsmatrix bestimmt. Um die Qualität des Hintergrunds im Vergleich zur ROI zu verringern, werden die Werte der Quantisierungsmatrix mit einem Skalierungsfaktor verrechnet. Um das Bild wieder dekodieren zu können, müssen die Skalierungsfaktoren bekannt sein, deswegen müssen diese im abgespeicherten Bild vorgehalten werden. Dies bedeutet auch, dass kein Standard JPEG-Decoder verwendet werden kann.(siehe Golner und Mikhael (2000))

4.2 ROI-Komprimierungskonzept 1: Variable Block Größen

Dieses Konzept baut auf dem Quadtree Algorithmus aus 4.1.2 auf. Die Tiefe des Baums wird durch die Anwesenheit der ROI bestimmt. Wenn ein Block in der ROI liegt wird er unterteilt und die entstandenen Blöcke werden erneut geprüft(siehe Abbildung 4.2). Die einzelnen Blöcke werden als eigenes JPEG-Bild komprimiert und in einer Datenstruktur abgelegt. Anhand der Blockgrößen wird die Kompressionsstärke festgelegt. Bei der Kompression werden die kleinsten Blöcke, welche die ROI enthalten, mit höchster Qualität komprimiert und die größeren Blocke in niedriger Qualität. Das Beispiel aus Abbildung 4.2 könnte auch in fünf Teile zerlegt und ohne Quadtreeaufteilung komprimiert werden. Die Aufteilung des Bildes, mithilfe des Quadtree Algorithmus hat jedoch den Vorteil, das die ROI eine beliebige Form annehmen kann.

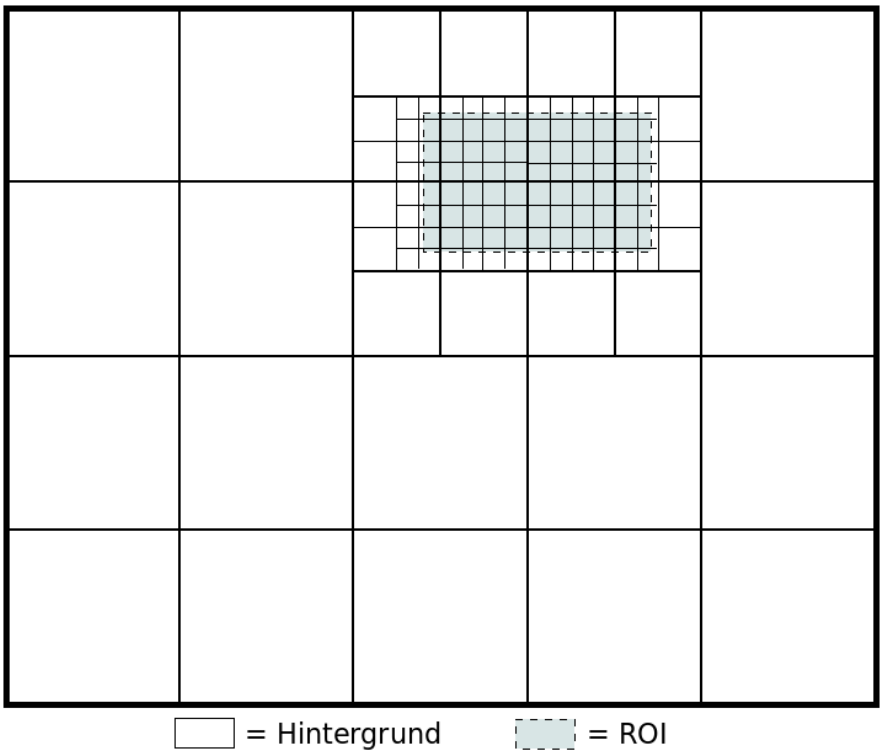

Abbildung 4.2: Anwendung des Quadtree Algorithmus, auf ein Bild mit ROI.

4.3 ROI-Komprimierungskonzept 2: Einfügen von Nullen in Quantisierten JPEG-Block

Das zweite Konzept greift ähnlich wie das Konzept zur Variablen Quantisierung aus Abschnitt 4.1.3, im Quantisierungsschritt des JPEG Algorithmus ein. Um die Komprimierung zu erhöhen, werden bei Blöcken, die nicht zu der ROI gehören, in den Quantisierten Block Nullen eingefügt. Eingefügt werden die Nullen am Ende des in Abschnitt 2.1.2 beschriebenen „Zickzackkurs“. Je mehr Nullen eingefügt werden, desto stärker kann die Lauflängencodierung die vorhandenen Daten komprimieren. In Abbildung 4.3 ist zu sehen, dass ab der vierten Wendung des Zickzackkurs nur noch Nullen eingelesen werden. Dadurch, dass als erstes die Werte der kleinen Frequenzen durch Nullen ersetzt werden, könnte es bei zu vielen Werten die ersetzt werden, zu Blockbildung kommen. Der Vorteil dieser Methode, im Vergleich zur Variablen Quantisierung ist jedoch, dass kein spezieller JPEG-Decoder benötigt wird. Jeder Standard JPEG Decoder kann die mit dieser Methode komprimierten Bilder decodieren.

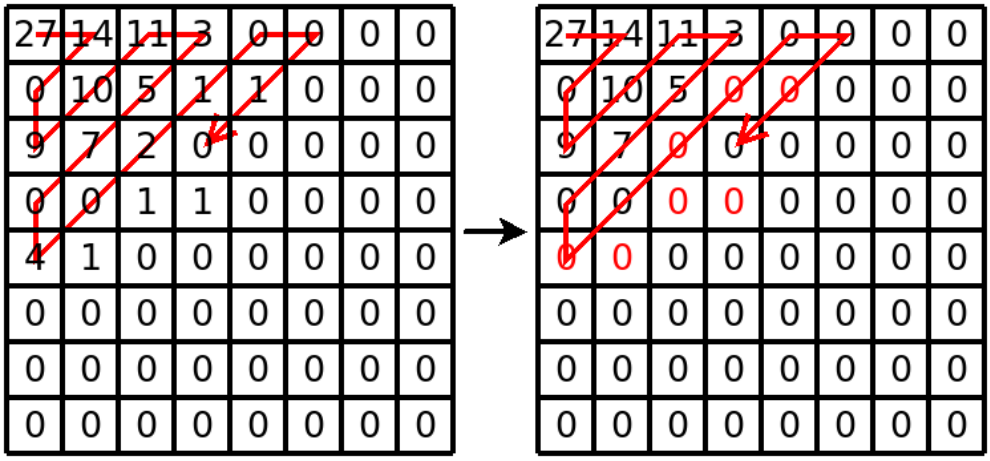

Abbildung 4.3: Beispiel für das Einfügen von Nullen in einen Quantisierten Block.

4.4 Umsetzung

In dieser Arbeit werden die Konzepte aus Abschnitt 4.2 und 4.3 beide umgesetzt. Danach sollen sie auf Kompressionsrate, die auf den Kameragateways erreichbare Framerate, sowie visuelle Qualität überprüft und miteinander verglichen werden. Es wir in dieser Arbeit nur mit rechteckigen ROIs gearbeitet.

4.5 Konzept zum Programmaufbau

In diesem Abschnitt wird das Konzept für den Aufbau des Hauptprogramms vorgestellt. Es soll folgendermaßen aufgebaut sein: Die zwei bestehenden Threads des Programms werden auf drei aufgeteilt, der Thread zum Versenden bleibt bestehen. Das Abrufen der Bilder von der Kamera und die Kompression werden jeweils von einem eigenen Thread ausgeführt. Die Threads sollen mit Hilfe von Mutexen und Semaphoren synchonisiert werden, anstatt mit Flags wie in der bisherigen Implementierung. Das Abrufen der Bilder im Aufnahmethread soll im Live Modus erfolgen, bei dem der Thread nicht wie im Snapshot Modus jedes Bild abfragen muss. Im Live Modus wird der Thread über ein Event darüber benachrichtigt, dass ein neues Bild vorhanden ist. In Abbildung 4.4 ist in einem Sequenzdiagramm der Programmablauf dargestellt. Der Kompressionsthread fragt vom Aufnahmethread das zuletzt aufgenommene Bild ab. Ist noch kein Bild vorhanden, blockiert der Kompressionsthread. Ist ein Bild vorhanden, wird das Bild mit dem ausgewählten Verfahren komprimiert. Der Versendethread ruft die

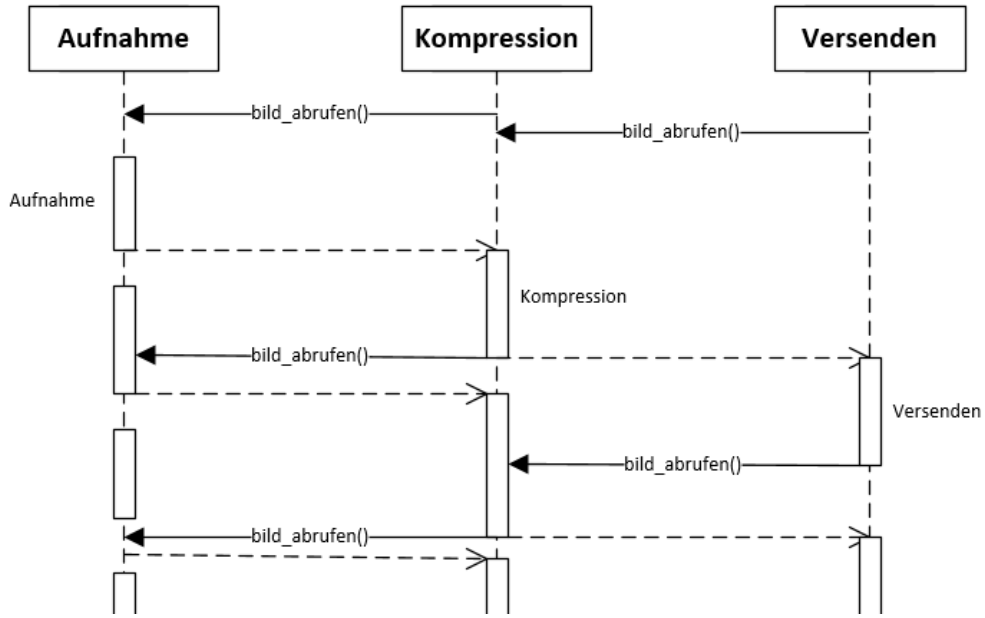

Abbildung 4.4: Sequenzdiagramm zum Programmablauf

Bilder vom Kompressionsthread nach dem gleichen Prinzip ab, wie der Kompressionsthread vom Aufnahmethread.

5 Implementierung

In diesem Kapitel werden die Details der Implementierung erläutert. Es wird die Umsetzung der zwei in Kapitel 4 vorgestellten ROI-Komprimierungskonzepte beschrieben, sowie der Aufbau und die Funktionsweise des gesamten Programms erklärt. Zusätzlich wird ein zu Testzwecken angefertigtes Empfangsprogramm erklärt und die für das einstellen der Kamera wichtigen Parameter aufgeführt.

5.1 ROI-Komprimierungskonzept 1

In diesem Abschnitt wird die Implementierung des in Abschnitt 4.2 beschriebenen Konzepts, im Detail erläutert. Es wird darauf eingegangen, wie das Bild in Blocke unterteilt wird, wie erkannt wird ob ein Block die Region of Interest enthält, wie die Blöcke abgespeichert werden und wie die Dekodierung funktioniert.

5.1.1 Aufteilung des Bildes in Blöcke

Das Aufteilen des Bildes findet in Blöcken der Größe 64x64, 32x32, 16x16 und 8x8 Pixel statt. Würde man die vier Größen kleiner wählen, würde zu viel Overhead erzeugt werden, da JPEG immer mit 8x8 Pixelblöcken arbeitet(siehe Abschnitt 2.1.2) und bei kleineren Einheiten padding anwendet. Würde man die vier Größen größer wählen, wären die Abweichungen des mit hoher Qualität komprimierten Bereichs zur ROI zu groß. Im Worstcase kann es dazu kommen, dass die ROI mit nur einem Pixel in einen 16x16 Block ragt und deswegen die vier daraus entstehenden Blöcke mit hoher Qualität komprimiert werden. Bei größeren Blockgrößen würde ein noch größerer Bereich, der eigentlich nicht zur ROI gehört, mit nicht benötigter hoher Qualität komprimiert. Zum Start der Kompression müssen einige Variablen initialisiert werden, um zu wissen, wo sich der aktuelle Block im Bild befindet, welche Größe der Block hat und wie viele Blöcke es gibt. Des Weiteren wird geprüft, welche Auflösung das Bild hat, da der Algorithmus nur Auflösungen verarbeiten kann, die ein Vielfaches von der größten Blöckgröße darstellen.

Der erste Schritt in der Verarbeitung ist eine Schleife, die das Bild in 64x64 Blöcke einteilt. Innerhalb der Schleife wird jeder dieser Blöcke mit dem Quadtree Algorithmus bearbeitet.

Zunächst wird ermittelt, ob der aktuelle Block Überschneidungen mit der ROI hat. Dies passiert indem geprüft wird, ob der Block außerhalb der ROI liegt. Es wird geprüft ob sich alle Eckkoordinaten des aktuellen Blocks über, unter oder neben der ROI befinden, da man bei dieser Prüfweise nur mit den Koordinaten von zwei Ecken rechnen muss. Würde man herausfinden wollen, ob sich ein Teil des Blocks innerhalb der ROI befindet, müsste man mit allen vier Eckkoordinaten des Blocks rechnen. Des Weiteren müssten Sonderfälle beachtet werden, wie z.B. dass sich die komplette ROI innerhalb eines Blockes befindet. Wird als Ergebnis bestimmt, dass sich der Block und die ROI überschneiden, unterteilt der Quadtree Algorithmus diesen Block. Die daraus entstandenen vier Blöcke werden rekursiv erneut behandelt, bis zu einer

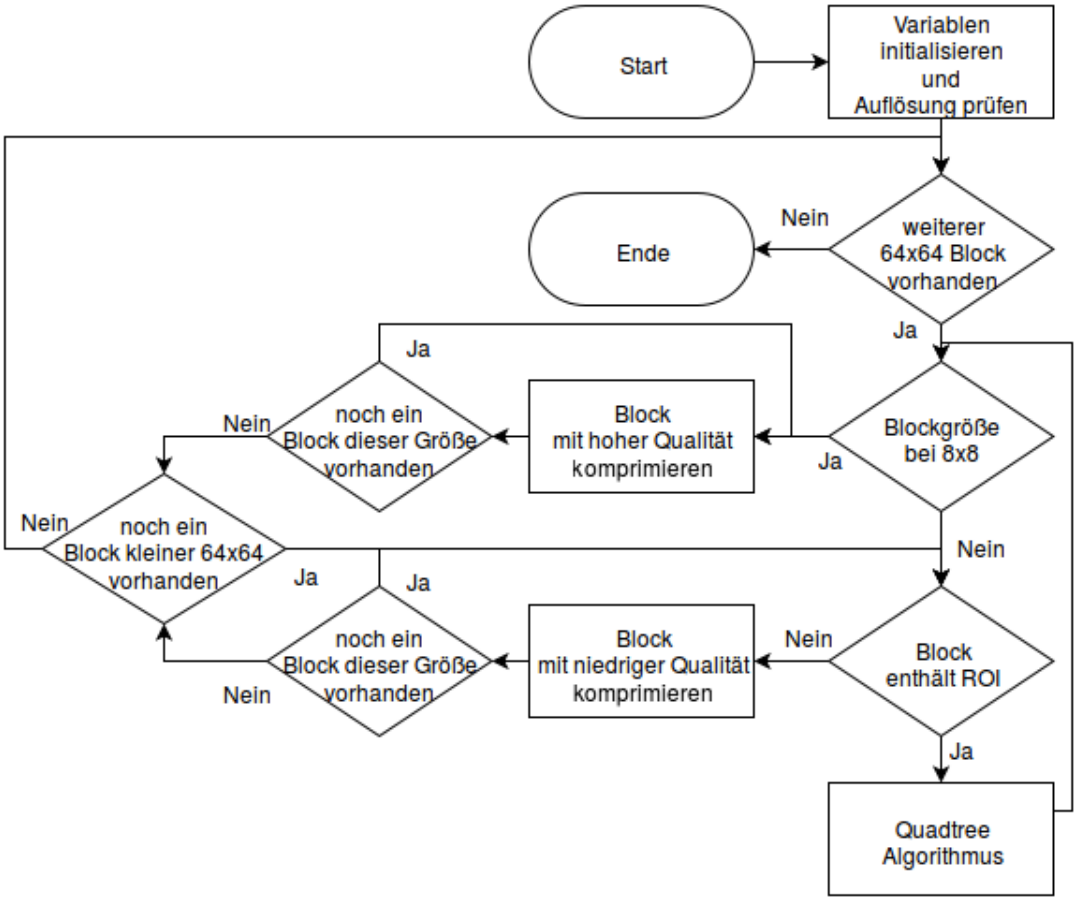

Abbildung 5.1: Ablaufdiagramm zur Unterteilung des Bildes in Blöcke.

minimalen Blockgröße von 8x8 Pixeln. Blöcke mit einer Größe von 8x8 Pixeln enthalten die ROI und werden als JPEG mit höchster Qualität komprimiert. Größere Blöcke die keine ROI enthalten, werden nicht weiter durch den Quadtree Algoritmuss behandelt und mit geringer

Qualität komprimiert. Jeder Block wird als eigenes JPEG komprimiert. Die Kompression mit Hilfe der libjpeg-turbo Bibliothek wird in Abschnitt 5.3.5 erklärt. In Abbildung 5.1 wird in einem Ablaufdiagramm, der genaue Ablauf der Unterteilung in Blöcke dargestellt.

5.1.2 Datenstruktur

Nachdem ein Block komprimiert wurde, wird er in einer Datenstruktur abgelegt. Diese Datenstruktur ist eine Klasse, die aus mehreren Objekten und Methoden aufgebaut ist. Sie ist in Abbildung 5.2 in einem Klassendiagramm dargestellt. Die als JPEG komprimierten Blöcke,

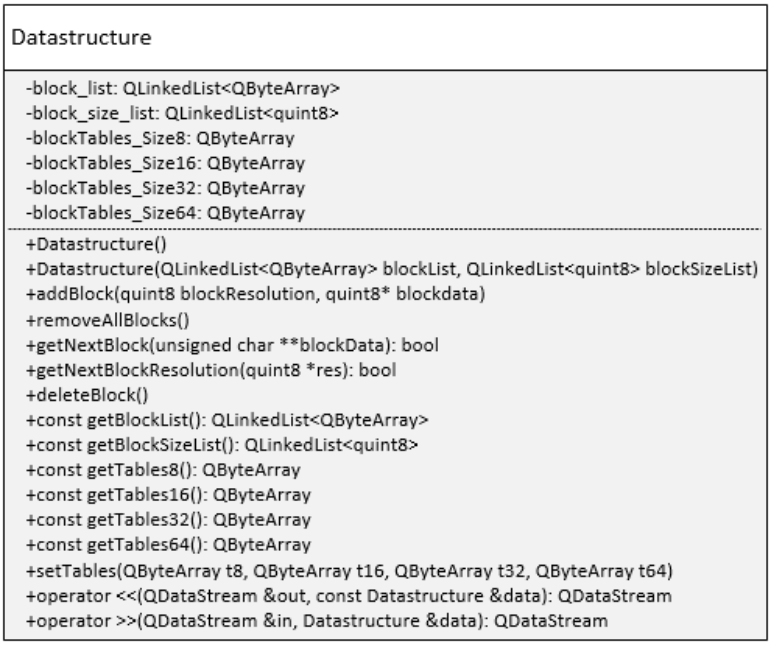

Abbildung 5.2: Klassendiagramm der Datenstruktur.

werden als Bytearray in einer verketteten Liste gespeichert. Zusätzlich gibt es eine Liste über Bytes, die die entsprechenden Blockgrößen speichert. Um unnötige Redundanzen in den JPEG Headern zu vermeiden, werden in den Headern der Blöcke die Quantisierungstabellen entfernt. Die Quantisierungstabellen sind bei zwei Bildern, die mit der gleichen Qualitätsstufe komprimiert wurden, gleich und werden deswegen für jede Blockgröße einmal separat in der Datenstruktur abgelegt. Die Klasse bietet Methoden, um auf die oben aufgeführten Elemente zuzugreifen. Außerdem ist sie serialisierbar, um über das Netzwerk verschickt werden zu können.

5.1.3 Dekomprimierung

Bei der Dekodierung wird über die Liste der Blöcke aus der Datenstruktur iteriert(siehe Abbildung 5.3). Zu jedem Block wir der passende Header, mit den passenden Quantisierungstabellen

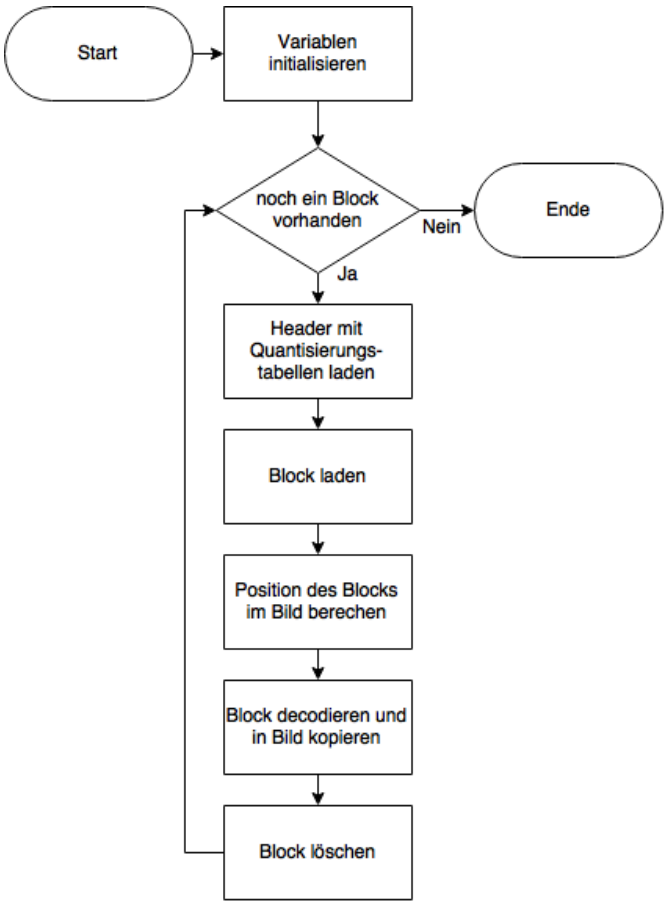

Abbildung 5.3: Decodieren der Blöcke aus der Datenstruktur.

eingelesen. Danach werden einige Variablen, die dazu dienen, den dekodierten Block an die richtige Stelle im fertigen Bild zu platzieren, aktualisiert. Dann wird der Block dekodiert und jede Pixelzeile an die entsprechende Stelle im vorher allozierten Bildspeicher geschrieben. Wenn ein Block dekodiert ist, wird er aus der Liste entfernt. Ist kein Block mehr vorhanden, ist das Bild komplett.

5.2 ROI-Komprimierungskonzept 2

In diesem Abschnitt wird die Implementierung des in Abschnitt 4.3 beschriebenen Konzepts erklärt. Für die Implementierung wurden Änderungen an der libjpeg-turbo Bibliothek vorgenommen, die Anpassungen am Algorithmus sowie an der API werden im Folgenden erläutert.

5.2.1 Erweiterungen der libjpeg-Turbo API

Die Standard libjpeg-turbo API wird nur um eine zusätzliche Funktion erweitert. Diese Funktion(siehe Listing 5.1) dient zum Übergeben der ROI, sowie zum Einschalten der ROI-Kompression.

```
1 EXTERN(boolean) jpeg_set_ROI(j_compress_ptr cinfo,
2                               unsigned int x_top,
3                               unsigned int x_bottom,
4                               unsigned int y_left,
5                               unsigned int y_right,
6                               unsigned int bg_quality);
```

Listing 5.1: Funktionsignatur zum Setzten der ROI in der libjpeg-turbo API.

Als Argumente werden die Kompressionsstruktur, die Koordinaten der ROI, sowie die Qualität des Hintergrunds übergeben. In der Funktion wird geprüft, ob die Werte der ROI und der Qualität valide sind. Das Ergebnis wird als Boolean zurückgegeben. Wenn die Werte korrekt sind, werden sie in der Kompressionsstruktur gespeichert. Zusätzlich erhält die Kompressionsstruktur ein Boolean, dass dem Rückgabewert der oben beschriebenen Funktion entspricht und worüber sich die ROI-Kompression ein- bzw. ausschalten lässt. Außerdem gibt es eine Variable, die während der Kompression eines Bildes die Zeilenanzahl zählt, um feststellen zu können, ob ein JPEG-Block in der ROI liegt. In Tabelle 5.1 sind alle zusätzlichen Variablen der Kom-

Parameter	Beschreibung
boolean roi_ compression	Schaltet ROI-Kompression einaus.
unsigned int roi_ x_ top	ROI Koordinate.
unsigned int roi_ x_ bottom	ROI Koordinate.
unsigned int roi_ y_ left	ROI Koordinate.
unsigned int roi_ y_ right	ROI Koordinate.
unsigned int roi_ row_ counter	Zeilenzähler zur ROI Erkennung.
unsigned int roi_ bg_ quality	Qualität des Hintergrunds

Tabelle 5.1: Zur Kompressionsstruktur hinzugefügt Variablen.

pressionsstruktur aufgeführt. Die Hintergrundqualität kann in 15 Stufen von 0-14 eingestellt werden, wobei 14 die höchste Qualität darstellt. Das gesamte Bild wird mit einem in Abschnitt 5.3.5 erläuterten JPEG-Qualitätswert komprimiert. Der Hintergrundqualitätswert bestimmt, wie stark die Qualität die durch den JPEG-Qualitätswert definiert wurde für den Hintergrund verringert wird. Angewendet wird die ROI-Kompression mit der gleichen Vorgehensweise wie bei einer normalen JPEG Kompression, die in Listing 5.4 in Abschnitt 5.3.5 zu sehen ist. Der einzige Unterschied besteht darin, dass nach dem setzen der Defaultwerte für das Bild, die oben aufgeführte Funktion aufgerufen werden muss.

5.2.2 Änderungen am Komprimierungsalgorithmus

Zum Ersetzen von Werten in einem quantisierten JPEG-Block, muss der Komprimierungsalgorithmus der Bibliothek abgeändert werden. Die Idee des Konzepts ist es, quantisierte Werte durch Nullen zu ersetzen. Aus Gründen der Performanz ist es jedoch sinnvoller, diese Werte nicht zu quantisieren und direkt durch Nullen zu ersetzen, um Rechenzeit zu sparen. Als erstes wird geprüft, ob die ROI-Komprimierung eingeschaltet ist. Ist dies der Fall, muss jeder Block bevor er quantisiert wird, darauf untersucht werden, ob er in der ROI liegt. Diese Überprüfung findet auf die gleiche Weise wie im ersten Konzept statt, es wird geprüft, ob der Block außerhalb der ROI liegt. Liegt ein Block in der ROI, wird dieser mit der normalen Quantisierungsfunktion quantisiert. Handelt es sich um einen Hintergrundblock, wird die abgeänderte Funktion genutzt, die in Listing 5.2 zu sehen ist.

```
1  void quantize_bg (JCOEFPTR coef_block, DCTELEM * divisors,
2                                         DCTELEM * workspace){
3    int i;
4    DCTELEM temp;
5    JCOEFPTR output_ptr = coef_block;
6    UDCTELEM recip, corr, shift;
7    UDCTELEM2 product;
8    /*for each element of the Block*/
9    for (i = 0; i < DCTSIZE2; i++) {
10     /*check if element can be replaced by zero*/
11     if((i%JPEG_BLOCK_SIZE)>
12         (((int)cinfo_p->roi_bg_quality)-(i/JPEG_BLOCK_SIZE))){
13       output_ptr[i] = (JCOEF) 0;//replace with zero
14     }else{
15       /*do regular quantisation from Library*/
16       ...
17     }
```

```
18     }
19 }
```

Listing 5.2: Funktion zum Quantisieren eines Hintergrund JPEG-Blocks.

In der Funktion wird in Zeile 11 für jeden Wert eines Blocks geprüft, ob dieser durch Null ersetzt werden kann oder ob eine normale Quantisierung durchgeführt werden muss. Ob ein Block ersetzt wird hängt von der eingestellten Qualität ab, die die Werte 0-14 haben kann. Wie sich die unterschiedlichen Qualitäten auf die Anzahl der eingefügten Nullen auswirken ist in Abbildung 5.4 zu sehen. Eine Qualität von Null bedeutet, dass nur der obere linke Wert

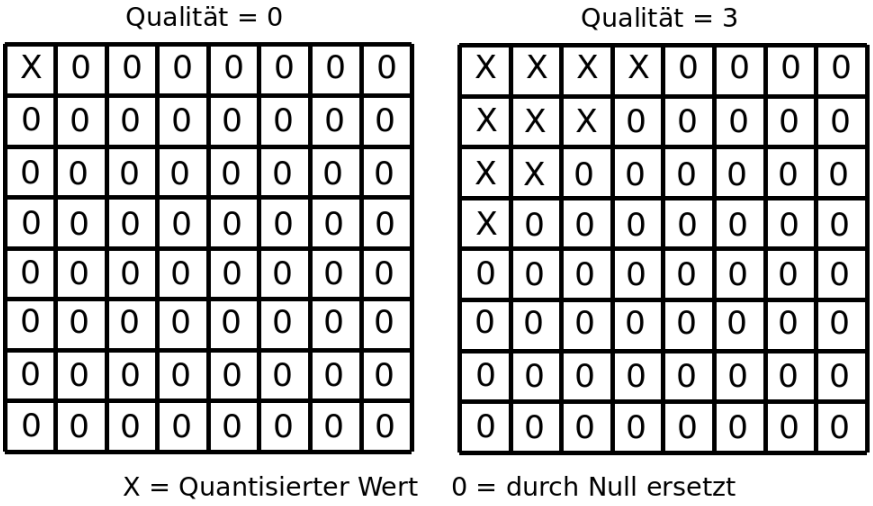

Abbildung 5.4: Anzahl der eingefügten Nullen bei unterschiedlichen Qualitäten.

des Blocks quantisiert wird. Bei jeder Erhöhung der Qualität, wird eine zusätzliche diagonale Reihe von Werten quantisiert. Die Reihen sind diagonal gewählt, da sie so durch die Zickzack-Umsortierung am Anfang stehen. Die Werte werden nicht in Zickzackreihenfolge quantisiert, sondern von links nach rechts und oben nach unten. Aufgrund dessen ist es einfacher und weniger rechenaufwendig, die Qualitätsabstufung in 15 diagonalen Reihen durchzuführen, als in 64 Werten.

5.3 Komprimierungsprogramm

In diesem Abschnitt wird der Programmaufbau des Komprimierungsprogramms erläutert. Dabei wird erklärt, wie die Bilder abgerufen, komprimiert, versendet und zwischen den Threads weitergeleitet werden.

5.3.1 Programmstruktur

Das Komprimierungsprogramm besteht neben der Main aus vier wichtigen Hauptklassen:

- Der *Camera* Klasse, über die mit der Kamera kommuniziert wird.
- Der *Capture* Klasse, die für das Abrufen der Bilder zuständig ist.
- Der *Compression* Klasse, die für die Kompression verantwortlich ist.
- Der *Transmit* Klasse, die das komprimierte Bild versendet.

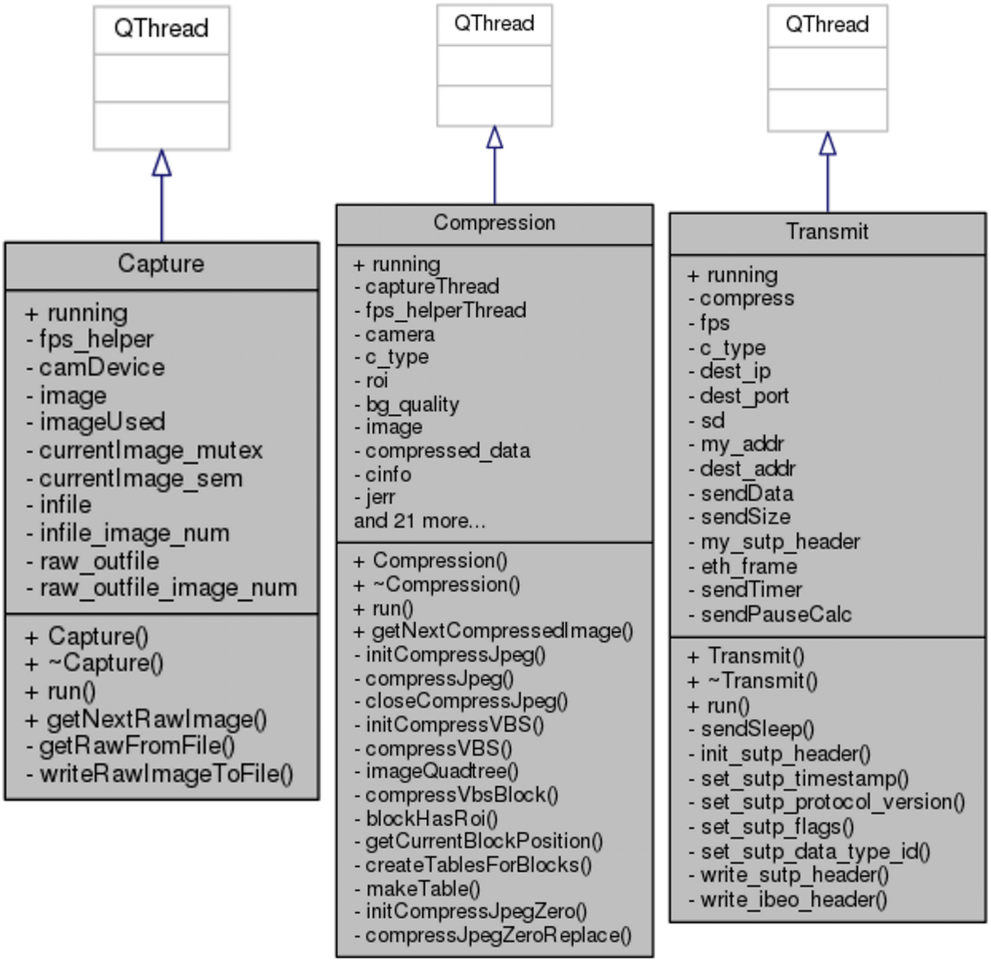

Abbildung 5.5: Klassendiagramm der Capture, Compression und Transmit Klasse.

Die letzten drei dieser vier Klassen erben, wie in Abbildung 5.5 zu sehen, von *QThread* und werden als Thread ausgeführt. Des Weiteren gibt es einen *FPS-Helper* Thread, die dem Benutzer

über die Kommandozeile die aktuelle Framerate ausgibt. Dieser und die *Camera* Klasse sind in Abbildung 5.6 zu sehen. Zusätzlich gibt es eine Klasse, die als Datenstruktur für Konzept 1

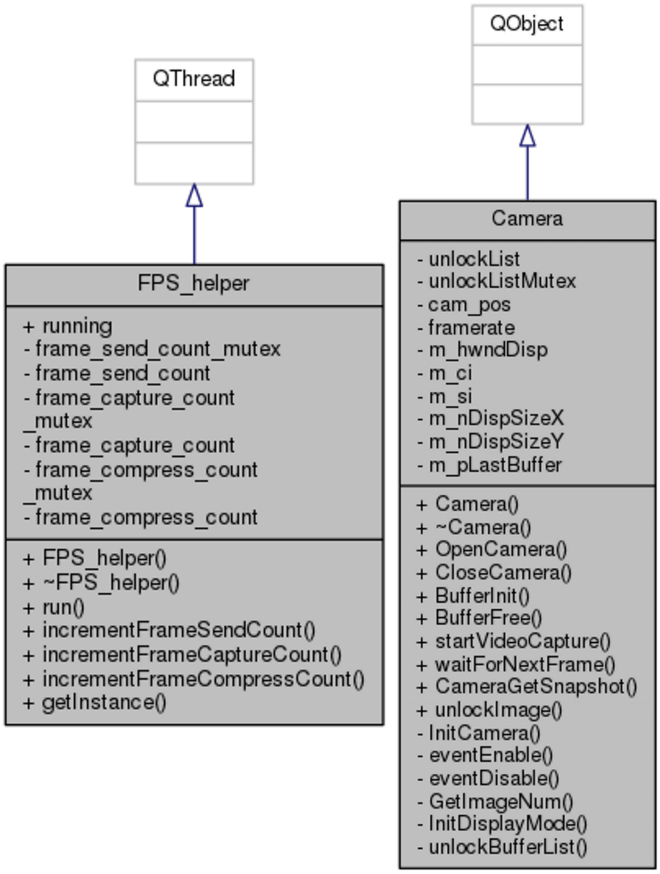

Abbildung 5.6: Klassendiagramm der FPS-Helper und Camera Klasse.

dient. Diese wurde in Abschnitt 5.1 genauer erläutert.

5.3.2 Übergabeparameter

Beim Start des Programms müssen dem Programm Parameter, die in Tabelle 5.2 aufgeführt sind, übergeben werden. Parameter Eins hilft dem Programm, die unterschiedlichen Parameter für die jeweilige Kamera einzustellen, die in Abschnitt 5.3.7 aufgeführt sind. Parameter Zwei gibt die vom Anwender gewünschte Framerate der Kamera an, die tatsächliche Framerate kann abweichen und wird dem Anwender nach dem Start mitgeteilt. Parameter Drei legt die für rate-constraint Traffic(siehe Abschnitt 2.4.1) nötige Pause zwischen zwei Ethernet-Frames und so auch die maximale Bandbreite der Anwendung fest. Parameter Vier legt das

Kompressionsverfahren fest und Parameter Fünf die Qualität des Hintergrundes bei den ROI Kompressionsverfahren. Parameter Sechs und Sieben geben IP und Port, des Ziels des Videostreams an. Die Parameter Acht bis Elf geben die rechteckige ROI an. Die ROI-Parameter müssen in der richtigen Reihenfolge angegeben werden und innerhalb des Bildes liegen, sie werden deswegen wie auch alle anderen Parameter geparst und überprüft.

Parameter	Wert	Beschreibung
1	FRONT/BACK	Teilt dem Programm mit, welche Kamera angeschlossen ist.
2	positive Kommazahl	Gewünschte Kameraframerate.
3	positive Ganzzahl	Pause zwischen dem Versenden von Ethernet-Frames in μs.
4	ZERO/VBS/NORMAL	Auswahl des ROI-Verfahrens bzw. normalem JPEG.
5	0-14 bzw. 1-100	Hintergrundqualität bei ROI Kompressionsverfahren(ZERO bzw. VBS).
6	gültige IPv4-Adresse	Zieladresse des Videostreams.
7	gültiger PORT	Zielport des Videostreams.
8	1-X	X-Wert, der oberen linken Ecke der ROI.
9	1-Y	Y-Wert, der oberen linken Ecke der ROI.
10	1-X	X-Wert, der unteren rechten Ecke der ROI.
11	1-Y	Y-Wert, der unteren rechten Ecke der ROI.

Tabelle 5.2: Auflistung der Übergabeparameter.

5.3.3 Programmstart und Programmablauf

Das Programm startet damit, die Übergabeparameter einzulesen und zu prüfen (siehe Abschnitt 5.3.2). Danach wird die Kamera, wie in Abschnitt 5.3.7 erläutert, initialisiert. Sind diese beiden Schritte erfolgreich abgeschlossen, werden die drei Threads zur Aufnahme, Kompression und zum Versenden erstellt und in dieser Reihenfolge gestartet. In Abbildung 5.7 ist ein Ablaufdiagramm des Programmstarts zu sehen.

Die drei Threads sind pipelineartig hintereinander geschaltet. Das Bild wird, wie im Sequenzdiagramm in Abbildung 4.4 dargestellt, vom Aufnahmethread, über den Kompressionsthread, bis zum Versendethread weitergereicht. Diese Struktur hat den Vorteil, dass mehrere Bilder zur gleichen Zeit verarbeitet werden können. Die Rohdaten des Bildes, werden von der Aufnahme bis zur Kompression als Pointer weitergegeben, um große Kopieroperationen zu vermeiden. Das kleinere, komprimierte Bild wird in den Versendethread kopiert, um während das Bild gesendet wird, den Kompressionsthread nicht zu blockieren.

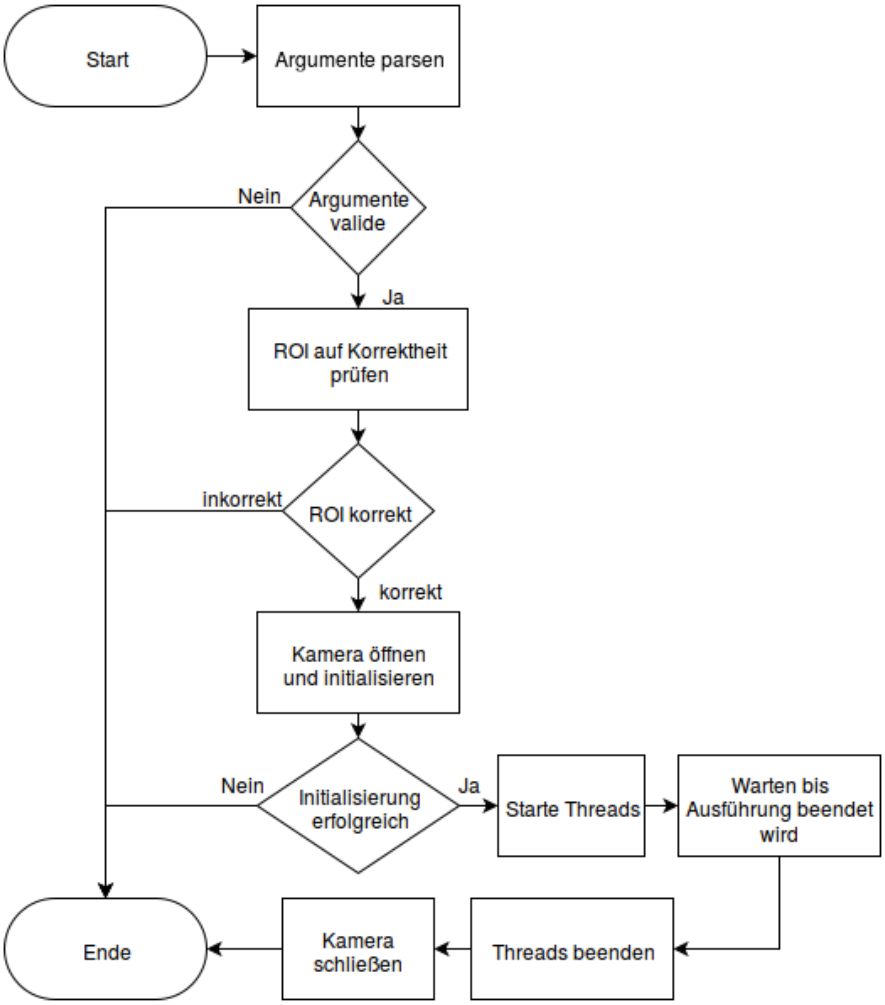

Abbildung 5.7: Ablaufdiagramm der Programm-main

5.3.4 Bildaufnahme

Nach erfolgreichem Start wird zunächst im Aufnahmethread der Live-Bildaufnahmemodus gestartet und auf ein Frame-Event der Kamera-API gewartet. Das Event signalisiert, dass ein neues Bild aufgenommen wurde. Die Bilder werden in einen für die Kamera angelegten Ringpuffer mit mehreren Bildspeichern aufgenommen. Ist dieses Event eingetroffen, wird der entsprechende Speicher des Ringpuffers gesperrt und die Adresse des Bildes kopiert und dem Kompressionsthread bereitgestellt. Der Ablauf des Aufnahmethreads ist in Abbildung 5.8 dargestellt. Der Kompressionsthread ruft das aufgenommene Bild über eine öffentliche Methode des Aufnahmethreads ab, die Synchronisation findet mithilfe eines Mutex und Semaphore statt.

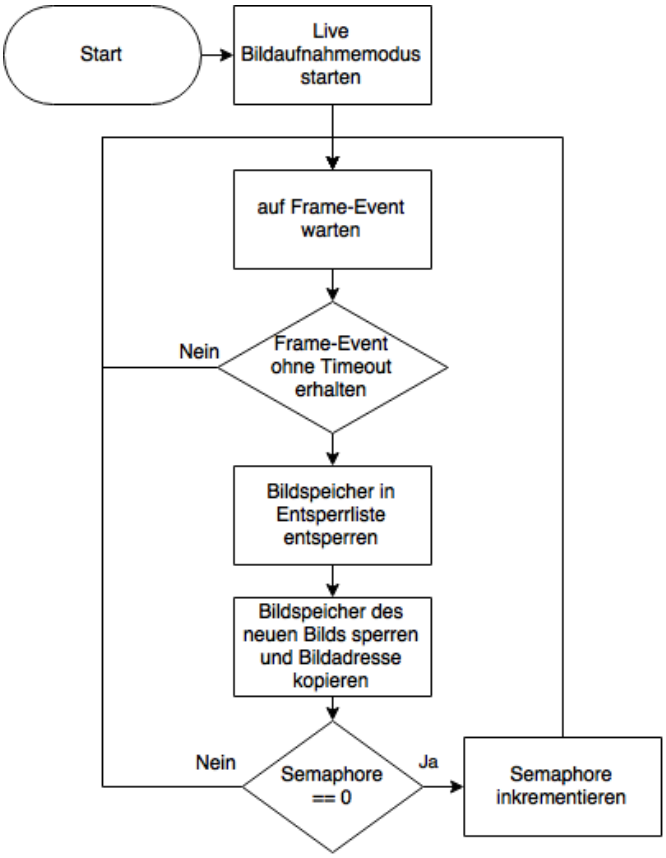

Abbildung 5.8: Ablaufdiagramm des Aufnahmethreads

Wird diese Methode aufgerufen, muss erst ein Semaphore reserviert werden. Dieser Semaphore wird bei einer erfolgreichen Bildaufnahme um Eins erhöht, erreicht jedoch nie einen Wert über Eins, da immer nur das aktuellste Bild vorgehalten wird. Wenn kein Bild vorhanden ist, blockiert der aufrufende Thread an dieser Stelle. Ist ein Bild vorhanden wird die Bildadresse vom Aufnahmethread zum Kompressionsthread weitergereicht.

Wenn ein Bild erfolgreich komprimiert wurde, wird die Adresse des Bildes im Ringpuffer vom Kompressionsthread in eine Liste geschrieben. Das Gleiche macht auch der Aufnahmethread mit einem alten Bild, wenn ein neues Bild eintrifft, dass Alte jedoch nicht abgerufen wurde. Mit Hilfe dieser Liste kann der Aufnahmethread, die nicht mehr für die Weiterverarbeitung benötigten Speicher des Ringpuffers, wieder entsperren.

5.3.5 Bildkompression

Der Kompressionsthread initialisiert nach der Erzeugung die für die gewählte ROI Kompressionsmethode benötigten Einstellungen, sowie die für alle Methoden benötigten JPEG Kompressionstrukturen. Dazu gehört eine Struktur, die die Kompressionsparameter beinhaltet und ein Error Handler. Des Weiteren wird, wie in Listing 5.3 zu sehen eingestellt, dass das komprimierte Bild im Hauptspeicher abgelegt werden soll.

```
//error and compression-info preparations
cinfo.err = jpeg_std_error(&jerr);
jpeg_create_compress(&cinfo);//create compression struct
pt_ImageData = NULL;//pointer to image data
ul_ImageSize = 0;//size of image
//set a memory destination for the image(instead of a file):
jpeg_mem_dest(&cinfo, &pt_ImageData, &ul_ImageSize);
```

Listing 5.3: Anlegen der JPEG Strukturen und einstellen des Speicherziels.

Nach dem Start wird zunächst die Adresse des nächsten Bilds vom Aufnahmethread abgerufen. Dieses Bild wird danach mit der eingestellten Kompressionsmethode weiterverarbeitet. Die Kompression eines Bildes, mit der libjpeg-turbo Bibliothek, findet wie in Listing 5.4 dargestellt statt.

```
void Compression::compressJpeg(quint8 *imagep){
    //set image parameters in compression struct
    cinfo.image_width = IMAGE_RES_X;
    cinfo.image_height = IMAGE_RES_Y;
    cinfo.input_components = IMAGE_COMPONENTS;
    cinfo.in_color_space = JCS_EXT_BGR;
    //set compression defaults, based on image info
    jpeg_set_defaults(&cinfo);
    //enable fastest DCT methode
    cinfo.dct_method = JDCT_FASTEST;
    //set JPEG-Quality Parameter
    jpeg_set_quality(&cinfo, CONVERT_QUALITY, TRUE);
    //set jpeg memory destination
    cinfo.dest->free_in_buffer +=
                cinfo.dest->next_output_byte - pt_ImageData;
    cinfo.dest->next_output_byte = pt_ImageData;
    // start compressor
    jpeg_start_compress(&cinfo, TRUE);
    //size of an pixel row in byte:
```

```
20      row_stride = IMAGE_RES_X * IMAGE_COMPONENTS;
21      //pass every pixel row to compressor
22      while (cinfo.next_scanline < cinfo.image_height){
23          row_pointer[0] = &(imagep[cinfo.next_scanline * row_stride]);
24          jpeg_write_scanlines(&cinfo, row_pointer, 1);
25      }
26      // finish compression
27      jpeg_finish_compress(&cinfo);
28  }
```

Listing 5.4: Kompression eines JPEGs.

Zuerst werden die Bildparameter in die Kompressionsstruktur geladen und die Standardwerte für die Kompression für diese Parameter gesetzt. Danach wird die schnellste DCT Methode gewählt, um den Rechenaufwand der Kompression zu reduzieren. Die DCT Methode muss nach dem setzen der Standardwerte erfolgen, da sonst die Standard DCT Methode genutz wird. Außerdem wird an dieser Stelle die Qualität des komprimierten Bildes festgelegt. Die Qualität wird als JPEG-Qualitätswert zwischen 0 und 100 angegeben, wobei 100 die beste Qualität darstellt. Bevor die Kompression gestartet wird, muss der Pointer auf die Zieladresse für das komprimierte Bild neu gesetzt werden, da es sonst bei Mehrfachverwendung der Kompressionsstruktur zu Speicherlecks kommt. Für die Kompression wird das Bild zeilenweise an den Kompressor übergeben. Ist die Kompression abgeschlossen, wird das Bild dem Versendethread zur Verfügung gestellt. Der Versendethread erhält das komprimierte Bild über eine öffentliche Methode des Kompressionsthreads. Auch die Interaktion dieser beiden Threads ist mit Hilfe eines Mutex und eines Semaphore gelöst. Wenn kein Bild vorhanden ist, blockiert der aufrufende Thread, indem er versucht ein Semaphore zu reservieren. Das Semaphore wird bei erfolgreicher Kompression freigegeben, jedoch nicht größer als Eins, da auch hier nur ein Bild zur Zeit vorgehalten wird. Da das komprimierte Bild wesentlich kleiner ist als das Originalbild, wird es kopiert und nicht als Pointer übergeben.

5.3.6 Netzwerkübertragung

Die Übertragung der Bilder im Netzwerk findet über UDP statt. Da es sich um ein Netzwerk mit TTEthernet Verkehr handelt und dieses Programm rate-constraint Traffic erzeugen soll, darf von der Anwendung eine bestimmte Bandbreite nicht überschritten werden. Dies wird erreicht, indem zwischen zwei Ethernetpaketen eine künstliche Pause eingelegt wird. Um diese Pause zu erzwingen, müssen die Bilddaten in Blöcke von der maximalen Größe eines Ethernetframes unterteilt werden, um dann einzeln nacheinander versendet werden zu können. Damit jeder

Ethernetframe eindeutig identifizierbar ist und im Sensorfusionsrechner verarbeitet werden kann, wird in die Payload des Frames noch ein zusätzlicher Header eingefügt. Der Header enthält Informationen darüber, in wie viele Ethernetframes das Bild aufgeteilt wurde und welche Nummer der aktuelle Ethernetframe hat. Außerdem sind die Sequenznummer des aktuellen Bildes und das Verfahren, mit welchem das Bild komprimiert wurde angegeben. Mit Hilfe dieses Headers kann der Empfänger ein fragmentiertes Bild wieder zusammensetzen und dekodieren. Das Senden eines Bilds im Versendethread findet wie in Abbildung 5.9 dargestellt statt. Nach dem Öffnen des UDP-Socket, wird das komprimierte Bild über den Kompressions-

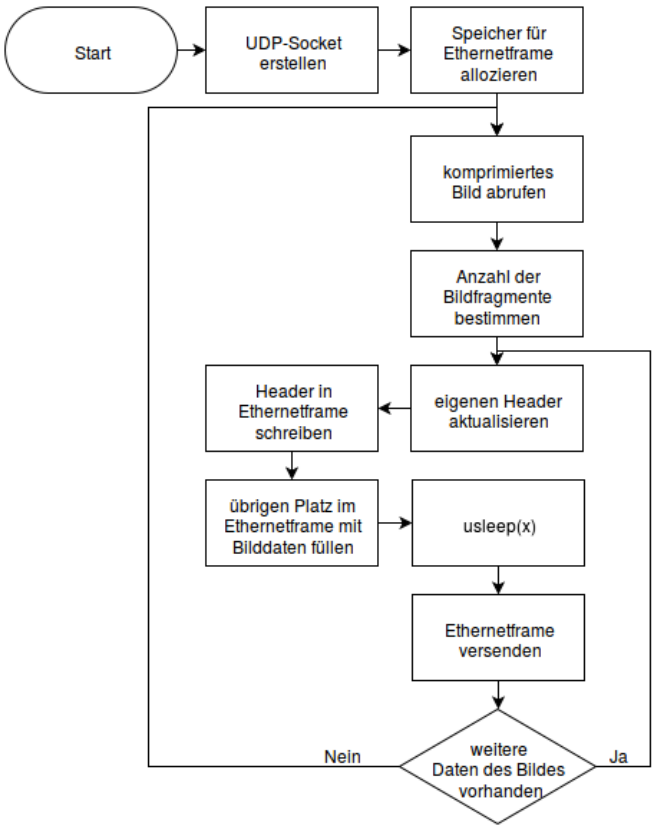

Abbildung 5.9: Ablaufdiagramm des Versendethreads.

thread abgerufen. Anhand der Datenmenge wird die Anzahl der benötigten Ethernetframes bestimmt. Danach wird der zusätzliche Header aktualisiert, in den Ethernetframe geschrieben und der übrige Platz des Frames mit Bilddaten gefüllt. Ist dies erfolgt wird der Frame versendet und die eingestellte Sendepause eingehalten. Sind noch Bilddaten vorhanden, wird ein weiterer Frame gesendet. Ist dies nicht der Fall, wird ein neues Bild vom Kompressionsthread abgerufen.

5.3.7 Kameraparameter

In diesem Abschnitt wird auf die unterschiedlichen Kameraparameter eingegangen, die eingestellt werden müssen. In der folgenden Tabelle 5.3 sind alle Kameraeinstellungen, die das Programm vornimmt, aufgelistet. Zuerst wird der Bilddarstellungsmodus gewählt. Dieser

Parameter	Modus/Wert vorderes Gateway	Modus/Wert hinteres Gateway
Bilddarstellungsmodus	Bitmap-Modus	Bitmap-Modus
Bildaufnahmemodus	Live/Videomodus	Live/Videomodus
Farbmodus	BGR8	BGR8
Belichtungszeit	Automatisch	Automatisch
Verstärkung	Automatisch	Automatisch
Verstärkung Referenz	80	80
Verstärkung Maximum	255	255
Pixeltakt	35MHz	25MHz
Kameraframerate	benutzerdefiniert	benutzerdefiniert

Tabelle 5.3: Auflistung der Kameraeinstellungen.

Modus bestimmt, wo die aufgenommenen Bilddaten abgelegt werden. Es gibt drei Modi, bei zwei dieser Modi werden die Bilder direkt im Grafikkartenspeicher abgelegt, um angezeigt zu werden. Da keine Bildanzeige, sondern eine Kompression stattfinden soll, wird der dritte Modus gewählt. Dieser sogenannte Bitmap-Modus schreibt die Bilddaten in den Hauptspeicher des Rechners. Nachdem der Bilddarstellungsmodus erfolgreich eingestellt worden ist, muss ein Farbmodus gewählt werden. Die wählbaren Farbmodi unterscheiden sich durch unterschiedliche Farbkanäle, sowie durch die Anzahl der Bit pro Farbkanal. Das für dieses Programm gewählte Standardformat BGR8, hat die drei Farbkanäle Blau, Grün und Rot, mit je 8 Bit Farbtiefe.

Als Nächstes muss der Pixeltakt so hoch wie möglich eingestellt werden, um hohe Bildwiederholraten an der Kamera einstellen zu können. Der Pixeltakt gibt an, mit welcher Geschwindigkeit die Pixel des Kamerasensors ausgelesen werden. Bei zu hoch eingestelltem Pixeltakt kann es bei der Übertragung der Bilder über USB, zwischen Kamera und Kameragateway, jedoch zu Übertragungsfehlern kommen. Dies hängt von der zur Verfügung stehenden Bandbreite der USB Verbindung ab. Aufgrund dessen kann am hinteren Kameragateway (siehe Abschnitt 2.2.3) nur ein Pixeltakt von 25 MHz eingestellt werden. Am vorderen Gateway kann der maximale Takt von 35 MHz eingestellt werden. Außerdem wird die Belichtungszeit und die Verstärkung in den Automatikmodus gesetzt, damit die Kamera die Helligkeit der Bilder automatisch an die äußeren Umstände anpasst und keine Über- bzw. Unterbelichtung stattfin-

det. Des Weiteren wird ein Referenzwert und eine Obergrenze für die Verstärkung festgelegt, diese legen die durchschnittliche und maximale Helligkeit der Bilder fest. In Abbildung 5.10

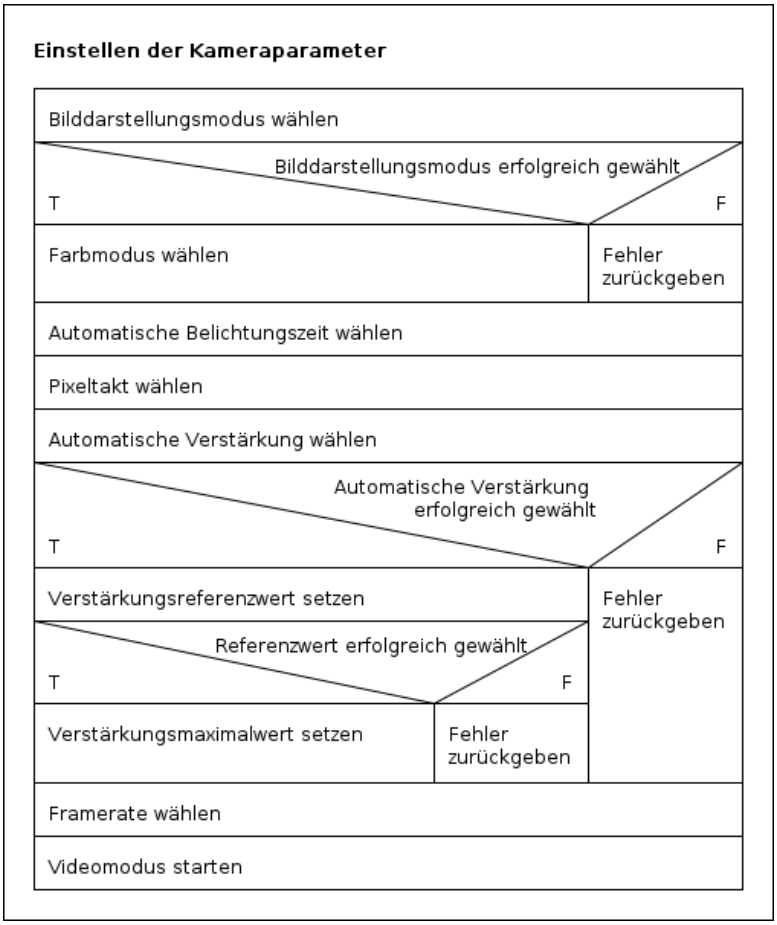

Abbildung 5.10: Diagramm zum einstellen der Kameraparameter.

ist der Ablauf der Einstellungen abgebildet. Die Einstellungen müssen in dieser Reihenfolge erfolgen, da z.B. der Farbmodus nur bei eingestelltem Bitmap-Modus gewählt werden kann und die Belichtungszeit vor dem Pixeltakt eingestellt werden muss. Beim einstellen der Framerate muss darauf geachtet werden, dass diese nicht höher eingestellt wird als die Rate, die das weiterverarbeitende Programm in der Lage ist pro Sekunde zu verarbeiten. Da die Kamera jedes nicht abgerufene Bild ca. eine Sekunde lang vorhält und immer das älteste Bild übergibt, entsteht bei zu hoch eingestellter Framerate, die in Abschnitt 3.3.1 beschriebene Verzögerung. Eine genaue Beschreibung der verschiedenen Modi und einstellbaren Parameter ist im Manual des Kameraherstellers zu finden(siehe (IDS Imaging, a)).

5.4 Empfangsprogramm

In diesem Abschnitt wird der Programmaufbau des Empfangsprogramms, das zum Überprüfen der Funktion des Komprimierungsprogramms entwickelt wurde, erläutert. Es wird kurz erklärt, wie die Bilder empfangen, angezeigt und zwischen den Threads weitergeleitet werden.

5.4.1 Programmstruktur und Programmablauf

Das Programm besteht aus drei nebenläufigen Threads.

- Der *Receive* Thread übernimmt das Empfangen der Bilder.
- Der *Decode* Thread ist für das Dekodieren der Bilder verantwortlich.
- Der *GUI* Thread ist für die Anzeige der Bilder zuständig.

Im Empfangsprogramm trifft ein Bild über das Netzwerk im *Receive* Thread ein. Hier wird mithilfe des in Abschnitt 5.3.6 beschriebenen Headers, das Bild wieder zusammengesetzt und erkannt, ob ein Fehler bei der Übertragung auftreten ist. Ist ein Bild komplett eingetroffen, wird es an den *Decode* Thread weitergeleitet. Zum Weiterreichen der Bilder werden Signale und Slots benutzt, ein Mechanismus der durch das Qt-Framework bereitgestellt wird. Mit Signalen und Slots können Objekte ereignisgesteuert miteinander kommunizieren. Dieser Mechanismus findet im Zusammenhang mit GUI-Programmierung häufig Verwendung. Der *Decode* Thread dekodiert das Bild nur, wenn es mit dem ersten ROI-Komprimierungskonzept(siehe Abschnitt 4.2) komprimiert wurde. Bei einem normalen JPEG bzw. einem mit dem zweiten ROI-Komprimierungskonzept komprimierten Bild, kann das Bild bei der Anzeige in der GUI direkt durch das Framework dekodiert werde. Die Funktionsweise der Dekodierung des ersten ROI-Komprimierungskonzepts wird in Abschnitt 5.1.3 erläutert.

5.4.2 GUI

Die GUI des Empfangsprogramm ist mithilfe des Qt-Frameworks erstellt worden. Sie dient nur zur Darstellung des aktuellen Bildes und ist deswegen sehr einfach gehalten(siehe Abbildung 5.11). Sie besteht aus einem *QDialog* Fenster, das eine *QGraphicsScene* enthält, auf der mithilfe einer *QPixmap* das Bild dargestellt wird. Ein neues Bild wird mittels eines Signals vom Dekodierthread an einen Slot der GUI geschickt. Das Signal enthält einen Pointer auf die Bilddaten, sowie die Größe des Bildes. Bei Eintreffen eines Signals wird die *QGraphicsScene* geleert und das neue Bild in die *QPixmap* geladen und wieder angezeigt.

Abbildung 5.11: Bild der GUI des Empfangsprogramms, mit Kamerabild.

6 Qualitätssicherung

Um die fehlerfreie Funktion des Programms sicherzustellen, wurden unterschiedliche Überprüfungen und Tests durchgeführt, die in diesem Kapitel erläutert werden.

6.1 Komponententests

Mit Komponententests kann die korrekte Funktionalität von einzelnen Funktionen und Methoden festgestellt werden. Im Rahmen dieser Arbeit wurden Komponententests für die Methoden der Datenstruktur von Konzept 1 erstellt, sowie für einige Methoden die bei der Kompression eine Rolle spielen. Die Kompression als solche kann jedoch nicht gut durch Komponententests getestet werden, da das komprimierte Bild kein leicht überprüfbarer Wert ist. Die Tests sind als Whitebox Test zu betrachten. Es wurden Tests durchgeführt, bei denen die getesteten Funktionen erfolgreich ihre Funktion ausführen, sowie Tests, bei denen die Funktionen einen Fehler zurückgeben sollen. Zudem wurden Grenzfälle berücksichtigt.

6.2 Überprüfen des Programms auf Speicherlecks

Zum Überprüfen des Programms auf Speicherlecks wurde die Valgrind Tool Suite (siehe Valgrind Developers) benutzt. Mit Hilfe des Valgrind Memcheck Tools können Speicherlecks gefunden und lokalisiert werden. Das Komprimierungsprogramm wurde sowohl bei der Komprimierung normaler JPEG Bilder, als auch bei der Anwendung der beiden ROI Komprimierungsverfahren überprüft. In Listing 6.1 ist ein Auszug der Ergebnisse des Tools zu sehen.

```
1 LEAK SUMMARY:
2    definitely lost: 0 bytes in 0 blocks
3    indirectly lost: 0 bytes in 0 blocks
4      possibly lost: 22,226 bytes in 323 blocks
5    still reachable: 22,550,901 bytes in 19,794 blocks
6         suppressed: 0 bytes in 0 blocks
```

Listing 6.1: Auszug aus den Ergebnissen des Valgrind Memcheck Tools.

Das Listing zeigt, dass es keine definitiv bzw. indirekt verlorenen Speicherbereiche gibt. Es gibt ca. 22kb, die möglicherweise verloren gegangen sind. Diese Bereiche haben ihren Ursprung allerdings in der libjpeg-turbo Bibliothek und werden bereits zum Start des Programms erzeugt. Da mit zunehmender Laufzeit jedoch keine Zunahme dieser Bereiche stattfindet, ist die Funktionalität des Programms nicht gefährdet.

6.3 Überprüfung der Funktionalität der Kompression und Netzwerkübertragung

Die funktionale Überprüfung der verschiedenen Kompressionsverfahren wurde durch visuelle Kontrolle der Bilder durchgeführt. Hierfür wurde das in Abschnitt 5.4 beschriebene Empfangsprogramm entwickelt, mit dem die komprimierten Bilder betrachtet werden können. Da das Empfangsprogramm die Bilder über das Netzwerk erhält, kann mit dem Programm auch die Funktionalität der Netzwerkübertragung getestet werden. Zusätzlich wurde die Netzwerkübertragung durch Mitschnitt und Inspektion des Netzwerkverkehrs mit Wireshark geprüft. In Abbildung 6.1 ist die von Wireshark gemessene Zeitdifferenz zwischen Ethernetframes zu

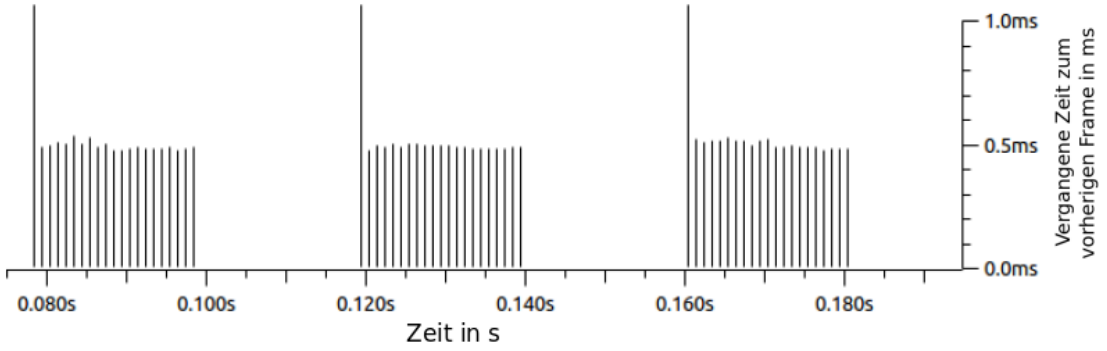

Abbildung 6.1: Zeitdifferenz zwischen zwei Ethernetframes.

erkennen. Um rate-constraint Traffic zu erzeugen, wurde in diesem Versuch eine Sendepause von 500 μs zwischen zwei Frames eingestellt. Im Diagramm sind drei Bilder zu erkennen, die mit einem Abstand von ca. 40ms (ca. 25fps) versendet werden. Des Weiteren ist zu sehen, dass die eingestellte Sendepause eingehalten wird.

7 Evaluierung/Ergebnisse

In diesem Kapitel werden die Ergebnisse der Arbeit ausgewertet. Die beiden umgesetzten Konzepte werden mit der Standard JPEG-Kompression verglichen. Sie werden anhand der Dateigröße, der erreichten Framerate auf einem Kameragateway sowie der visuellen Qualität verglichen. Die aus den Vergleichen erhaltenen Daten werden analysiert und bewertet.

7.1 Vergleich der Kompressionsverfahren

Für den Vergleich der Kompressionsverfahren anhand der Dateigröße, werden drei Beispielbilder verwendet, die bei Fahrten mit dem RECBAR Versuchsfahrzeug aufgenommen wurden. Die Bilder sind so gewählt, dass sie unterschiedlich viele Details haben und so schon durch normale JPEG Kompression unterschiedlich groß sind. Das dient dazu, die Kompressionsverfahren bei unterschiedlich detailreichen Bildern zu testen. In Abbildung 7.1 sind die drei Testbilder, sowie das Kamerabild, das während der Frameratenmessung genutzt wurde, abgebildet. Für den Vergleich mit den ROI Kompressionsverfahren werden die Bilder als JPEG mit einer JPEG Qualität von 95 komprimiert. Bei Bildern die mit den ROI Verfahren komprimiert werden, hat die ROI auch diese Qualitätsstufe. Diese Qualitätsstufe wurde gewählt, da höhere Werte die Dateigröße stark erhöhen, jedoch so gut wie keine sichtbaren Verbesserungen bei der Bildqualität bringen(siehe libjpeg-turbo (a)). In Tabelle 7.1 sind weitere Infos zu den Testbildern zusammengefasst.

Bild	Name	Auflösung	JPEG Dateigröße	Originalgröße
(a)	Autobahn	1280x1024	204,1 kB	3932,2kB
(b)	Ampel	1280x1024	298,9 kB	3932,2kB
(c)	Seitenstraße	1280x1024	364,8 kB	3932,2kB
(d)	Garage(Testbild Framerate)	1280x1024	340,1 kB	3932,2kB

Tabelle 7.1: Infos der Testbilder.

Bei dem Vergleich der Dateigröße und der Framerate, müssen zwei verschiedene Variablen berücksichtigt werden. Die Größe der ROI, sowie die Qualität des Hintergrundes (Bereich

Abbildung 7.1: Testbilder für den Vergleich der Kompressionsverfahren.

außerhalb der ROI). Es werden jeweils zwei Vergleiche der Dateigröße und der Framerate durchgeführt:

- Vergleich der Dateigröße/Framerate, bei Veränderung der Hintergrundqualität und gleichbleibend großer ROI.
- Vergleich der Dateigröße/Framerate, bei Veränderung der ROI Größe und gleichbleibender Hintergrundqualität.

Zusätzlich werden die Hintergründe bei verschiedenen Qualitätseinstellungen miteinander verglichen. Die Frameratenmessungen wurden auf dem Front-Kameragateway des Versuchsfahrzeugs durchgeführt. Für die Frameratenmessung wurde nur ein Bild verwendet, da hier der Inhalt des Bildes keine Auswirkungen hat.

7.1.1 Vergleich der visuellen Qualität des Hintergrundes

In diesem Abschnitt werden zwei Bildausschnitte der Testbilder in unterschiedlichen Qualitäten komprimiert und verglichen. Es wird ein Ausschnitt mit vielen Details und ein Ausschnitt mit wenigen Details gewählt. Da die Hintergrundqualität bei beiden ROI Kompressionsverfahren nicht durch einen einheitlichen Parameter festgelegt wird, kann jedoch kein direkter Vergleich zwischen zwei mit gleichem Qualitätswert komprimierten Ausschnitten gemacht werden. In Abbildung 7.2 ist ein Ausschnitt mit vielen Details zu sehen, in Abbildung 7.3 ein Ausschnitt mit wenigen Details. Zusätzlich zu den Ausschnitten der ROI Kompressionsverfahren, ist

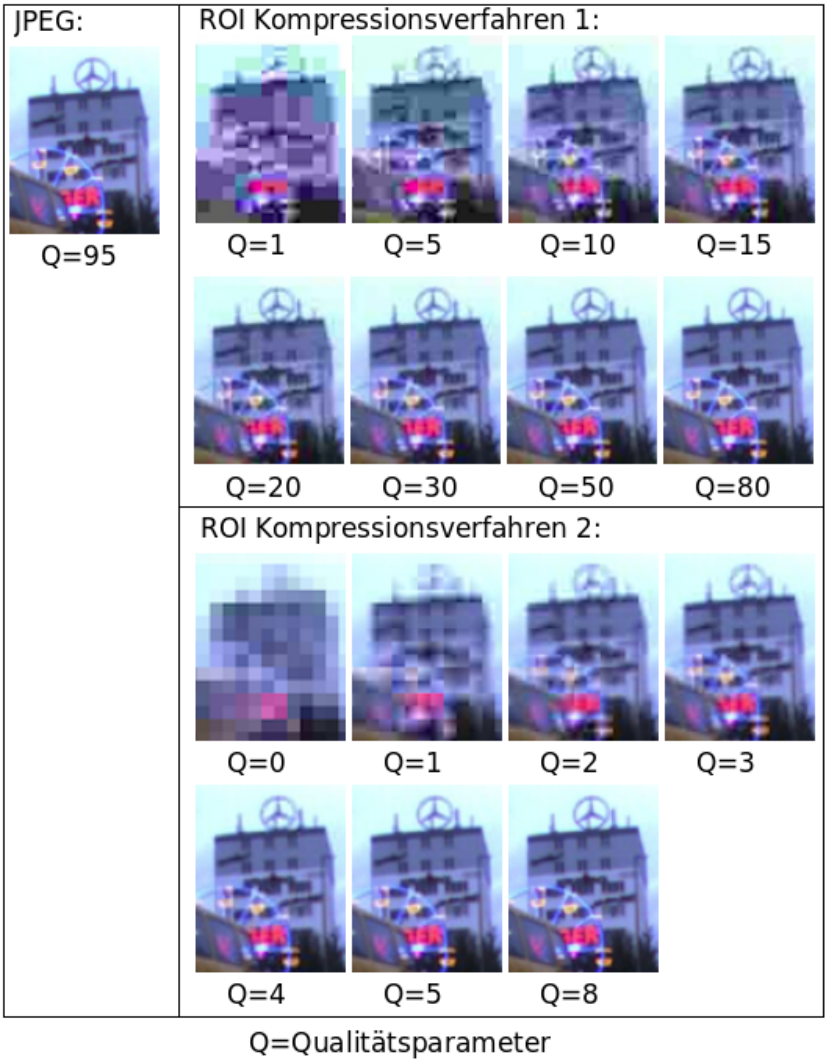

Abbildung 7.2: Vergleich der visuellen Qualität, anhand eines Bildausschnitts mit vielen Details.

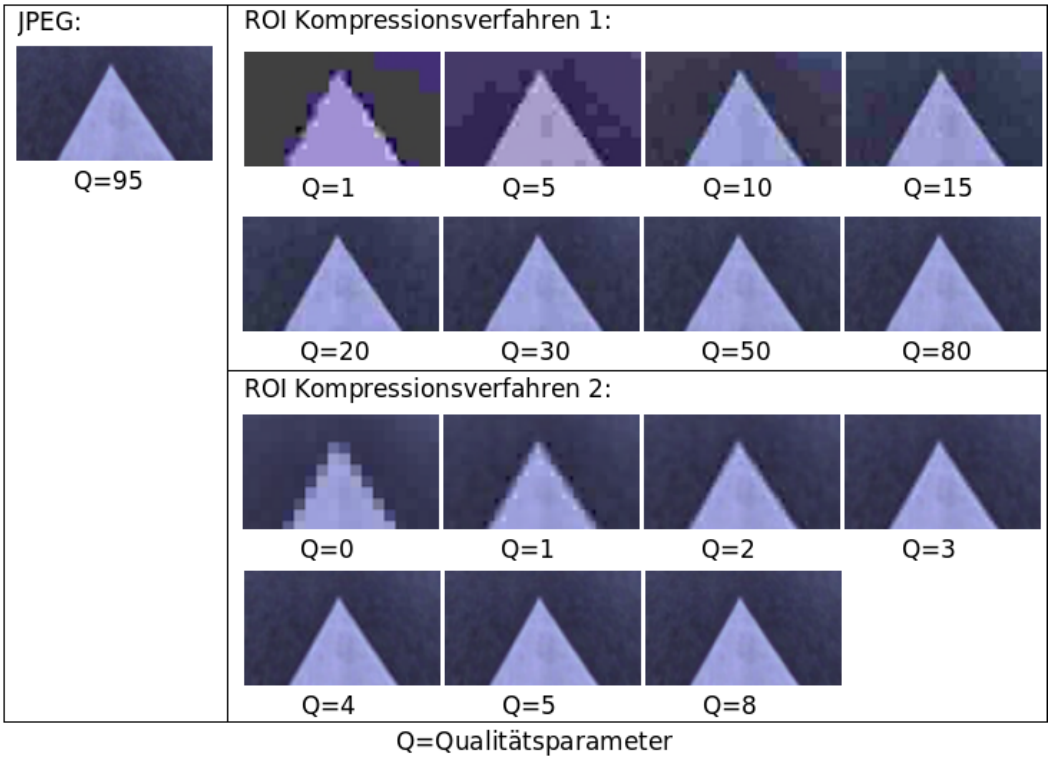

Abbildung 7.3: Vergleich der visuellen Qualität, anhand eines Bildausschnitts mit wenig Details.

in jeder Abbildung der gleiche Ausschnitt als JPEG zu sehen, wie er in der ROI aussehen würde. Es ist zu erkennen, dass bei beiden Kompressionsverfahren die visuelle Qualität im niedrigen Wertebereich des Qualitätsparameters stark zunimmt. Im höheren Wertebereich ist kaum noch visuelle Veränderung festzustellen. Außerdem ist zu erkennen, dass bei ROI Verfahren eins, Details und Kanten auch bei niedrigen Qualitätswerten deutlicher zu erkennen sind, als bei Verfahren zwei. Dies liegt daran, dass in Verfahren zwei bei niedrigen Qualitäten, die Informationen der detaillierten Bereiche durch Nullen ersetzt werden. Der Nachteil von Verfahren eins bei niedrigen Qualitätswerten ist, dass die dargestellten Farben stärker vom Original abweichen.
In den Abbildungen 7.5 und 7.4 sind beispielhaft zwei Testbilder mit einer ROI abgebildet. Die Bilder haben eine ROI die 5% des Bildes abdeckt und den Bildausschnitt, der in Fahrtrichtung liegt belegt. Um die ROI in den Bildern gut sichtbar zu machen, wurde die niedrigste Hintergrundqualität eingestellt.

Abbildung 7.4: Beispielbild von ROI Kompressionsverfahren Eins mit einer ROI Größe von 5% und niedrigster Hintergrundqualität.

Abbildung 7.5: Beispielbild von ROI Kompressionsverfahren Zwei mit einer ROI Größe von 5% und niedrigster Hintergrundqualität.

7.1.2 Vergleich der Dateigröße/Framerate, bei Veränderung der ROI Größe

Um die Dateigröße und Framerate bei Veränderung der ROI zu vergleichen, muss ein fester Qualitätswert für den Hintergrund festgelegt werden. Da es, wie in Abschnitt 7.1.1 beschrieben, keinen einheitlichen Parameter gibt der die Hintergrundqualität festgelegt, muss für jedes ROI Kompressionsverfahren ein eigener Qualitätswert festgelegt werden. Diese zwei Qualitätswerte werden so gewählt, dass die visuelle Qualität der Bilder ungefähr gleich ist. Für das erste ROI Verfahren wurde eine JPEG Qualität von 15 für den Hintergrund gewählt. Für das zweite ROI Verfahren eine Hintergrundqualität von Zwei. Beide Verfahren haben für die ROI eine JPEG Qualität von 95. Die ROI hat für die Messungen Größen von 1-50% des Bildes und liegt in der Mitte des Bildes. In Abbildung 7.6 sind die Ergebnisse der Messung der Dateigröße zu sehen. Es sind die Dateigrößen beider ROI Verfahren, in Abhängigkeit der ROI Größe, sowie

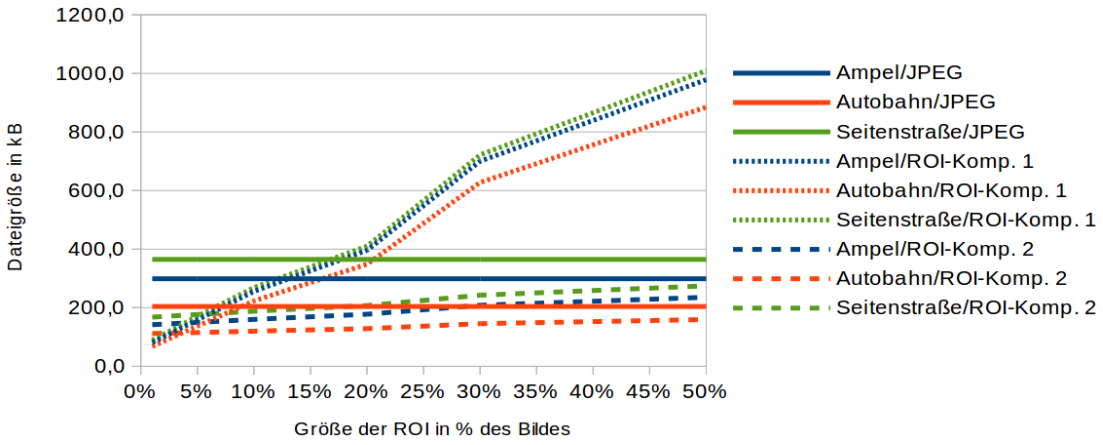

Abbildung 7.6: Dateigröße in Abhängigkeit der Größe der ROI.

die Dateigröße des Standard JPEGs zu sehen. Es zeigte sich, dass die Datenmenge bei dem zweiten ROI Verfahren, bei der kleinsten ROI Größe, in etwa halbiert wird. Danach steigt sie linear an und nähert sich der JPEG Dateigröße. Die Datenmenge des ersten ROI Verfahrens ist bei kleinen ROIs am geringsten, steigt jedoch schnell an und wird ab einer bestimmten ROI Größe sogar größer als die des JPEGs. Dies kann dadurch erklärt werden, dass durch den Anstieg der Anzahl der Blöcke die JPEG Header, trotz entfernen der Quantisierungstabellen, zu viel Datenoverhead erzeugen. Des Weiteren verlaufen die Kurven des zweiten Verfahrens nicht linear. Da die ROI, wie in Abschnitt 5.1.1 beschrieben, unterschiedlich weit in einen Block hereinragen kann, ändert sich die Datenmenge erst, wenn die ROI in einen weiteren Block hineinragt. Dadurch erhöht sich die Datenmenge eher sprungartig.
Abbildung 7.7 zeigt die Entwicklung der Framerate bei Vergrößerung der ROI. Es ist die Frame-

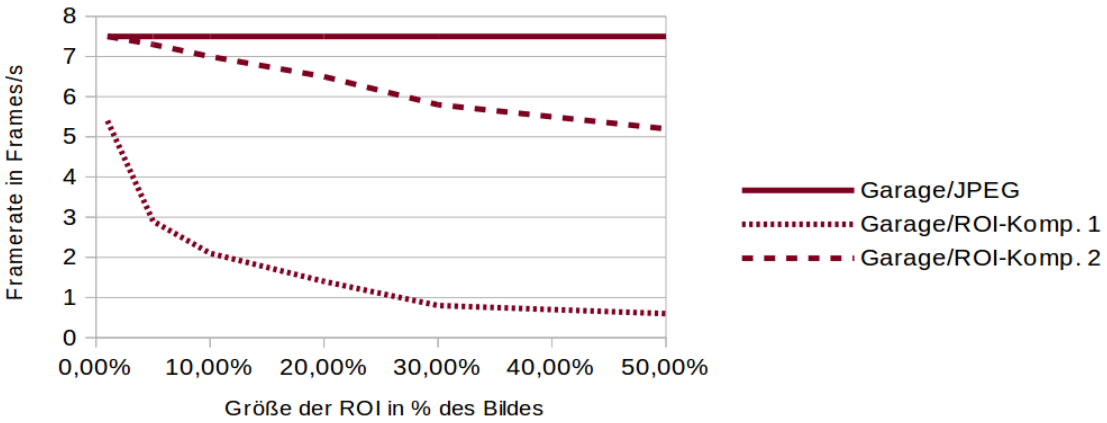

Abbildung 7.7: Framerate in Abhängigkeit der Größe der ROI.

rate der zwei ROI Verfahren, sowie des Standard JPEGs zu sehen. Zu erkennen ist, dass bei beiden ROI Verfahren die Framerate bei größeren ROIs abnimmt. Bei ROI Kompressionsverfahren Eins, ist die Framerate deutlich geringer als bei den anderen Varianten und bricht bei größeren ROIs deutlich stärker ein, als bei ROI Verfahren Zwei. Dies ist dadurch zu erklären, dass die Anzahl der Blöcke steigt und der gesamte Rechenaufwand für ein Bild deutlich zunimmt. In ROI Verfahren Zwei nimmt die Framerate weniger stark ab, da hier nur zusätzliche Quantisierungen durchgeführt werden müssen.

7.1.3 Vergleich der Dateigröße/Framerate, bei Veränderung der Hintergrundqualität

Für den Vergleich der Verfahren bei Veränderung der Hintergrundqualität, wird eine ROI Größe von 5% des Bildes verwendet. Es wird eine kleine ROI Größe gewählt, um einen prozentual größeren Einfluss der Qualität auf die Dateigröße/Framerate zu erhalten.
Für den Vergleich der Dateigrößen werden die beiden ROI Komprimierungsverfahren jeweils mit den Dateigrößen der Standard JPEG Kompression verglichen. In Abbildung 7.8 ist zu sehen, dass die Datenmenge bei ROI Verfahren Eins zunächst linear, und ab einer JPEG Qualität von ca. 75 exponentiell, ansteigt. Dies liegt an den Quantisierungstabellen, die dem entsprechenden Qualitätswert in der libjpeg-turbo Bibliothek zugeordnet sind. In Abbildung 7.9 ist zu sehen, dass bei ROI Verfahren Zwei die Datenmenge bei steigender Hintergrundqualität zunächst stärker ansteigt und dann abflacht. Dies hat zwei Gründe. Der erste ist, dass ab einer Hintergrundqualitätseinstellung von Sechs und höher, hauptsächlich Werte die in den meisten Fällen Null sind zusätzlich quantisiert werden. Der zweite Grund ist, dass die nicht durch Null

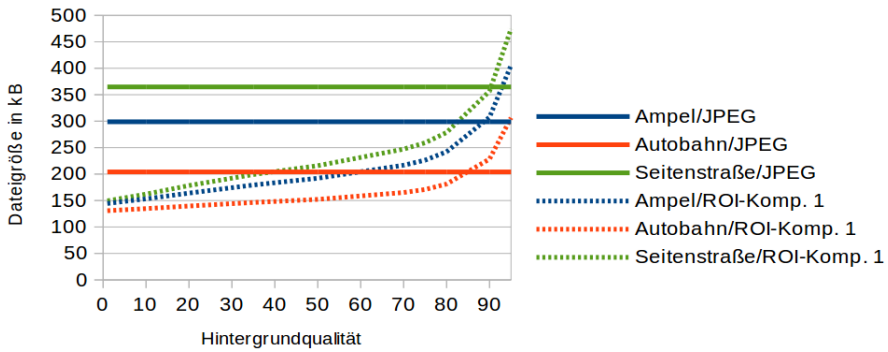

Abbildung 7.8: Dateigröße in Abhängigkeit der Hintergrundqualität(ROI Kompressionsverfahren 1).

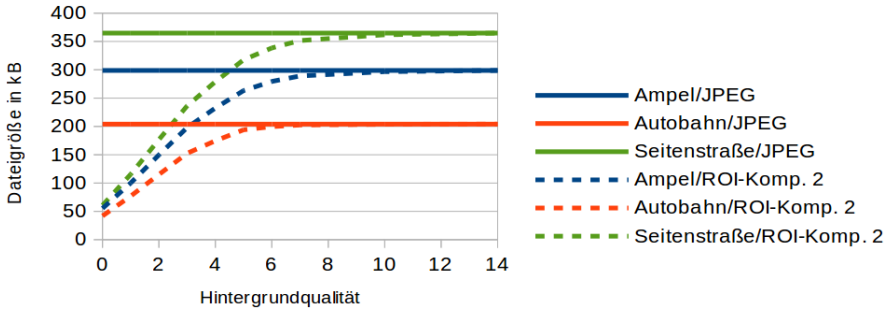

Abbildung 7.9: Dateigröße in Abhängigkeit der Hintergrundqualität(ROI Kompressionsverfahren 2).

ersetzten Werte nicht gleichmäßig zunehmen, wie in Abbildung 5.4 gezeigt.
Abbildung 7.10 und Abbildung 7.11 zeigen den Verlauf der Framerate bei Verbesserung der Hintergrundqualität. Es ist zu erkennen, dass bei ROI Verfahren Eins mit Veränderung der Hintergrundqualität, nur geringe Veränderungen bei der Framerate zu erkennen sind. Bei ROI Verfahren Zwei ist dies nicht der Fall. Die Framerate ist bei geringer Qualität höher als die bei normalen JPEGs, fällt dann jedoch ab. Dies ist dadurch zu erklären, dass im Gegensatz zu Verfahren Eins bei erhöhter Qualität mehr Rechenoperationen durchgeführt werden müssen. Es müssen mehr Werte quantisiert werden. Im Vergleich zum normalen JPEG Verfahren müssen zwar deutlich weniger Quantisierungen durchgeführt werden, jedoch wird dieser Performancevorteil ab einem Hintergrundqualitätswert von Zwei, durch die benötigte ROI Überprüfung für jeden Block wieder zunichte gemacht.

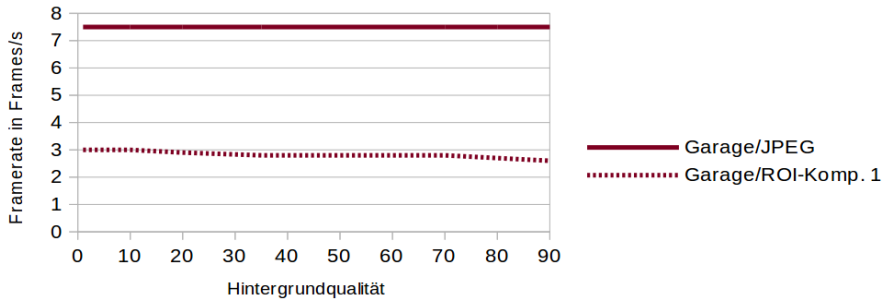

Abbildung 7.10: Framerate in Abhängigkeit der Hintergrundqualität(ROI Kompressionsverfahren 1).

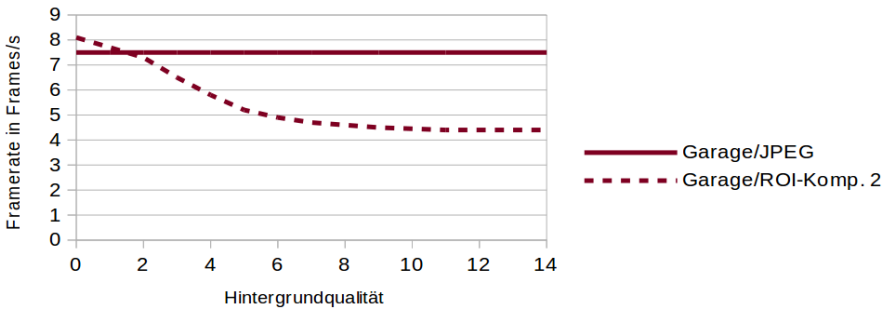

Abbildung 7.11: Framerate in Abhängigkeit der Hintergrundqualität(ROI Kompressionsverfahren 2).

7.2 Ergebnisbewertung

In den in diesem Kapitel dargelegten Vergleichen, sind die unterschiedlichen Eigenschaften der Kompressionsverfahren deutlich geworden. Das erste ROI Kompressionsverfahren reduziert zwar bei kleinen ROIs die Datenmenge stark, wird allerdings bei etwas größeren ROIs sowohl im Bereich der Framerate, als auch der Datenmenge unbrauchbar. Durch die Hintergrundqualität wird das Verfahren vergleichsweise wenig beeinflusst, diese kann deshalb bei einer Verwendung des Verfahrens recht hoch gewählt werden. Das zweite ROI Verfahren ist außer bei recht kleinen ROIs, was die Datenmenge betrifft, dass geeignetere Verfahren. Wenn die Framerate betrachtet wird, ist das zweite Verfahren dem ersten bei jeder möglichen Einstellung überlegen. Die eingestellte Hintergrundqualität wirkt sich jedoch wesentlich stärker aus, als die ROI Größe, sowohl auf Framerate als auch auf die Datenmenge. Deshalb muss bei der Verwendung dieses Verfahrens eine genaue Abwägung zwischen Hintergrundqualität und Datenmenge bzw. Framerate gemacht werden. Aufgrund dieser Ergebnisse, ist für den Anwendungsfall dieser

Arbeit und dessen Anforderungen, das zweite Verfahren das geeignetere.
Die in Abschnitt 3.1 aufgeführten Ziele und Anforderungen wurden teilweise erreicht.

- Eine möglichst gleichbleibende Netzwerklast ist durch Nutzung des JPEG Kompressionsverfahrens gegeben, und die von der Anwendung benötigte Bandbreite, kann durch die Verwendung einer ROI verringert werden.
- Eine Erhöhung der Framerate konnte auf der zur Verfügung stehenden Hardware nur bedingt erreicht werden. Die Framerate konnte durch einige Verbesserungen im Programm erhöht werden, jedoch wurde durch die Erhöhung der Auflösung und damit die Verdopplung der Rohdatenmenge, die maximale Framerate wieder reduziert. Deswegen wird im finalen Programm keine höhere Framerate erreicht. Dies liegt daran, dass die Hardware bei dieser Framerate komplett ausgelastet ist.
- Das in Abschnitt 3.3.1 beschriebene Problem der Latenz zwischen Aufnahme und Anzeige des Bildes, konnte wie in Abschnitt 5.3.7 beschrieben, durch Anpassung der Framerate behoben werden.

8 Fazit und Ausblick

In dieser Arbeit wurden zwei Konzepte zur ROI basierten selektiven Kompression entworfen und umgesetzt. Die zwei Konzepte wurden aufbauend auf dem JPEG Algorithmus entwickelt. Dieser Algorithmus hat sich für die Anforderungen des im Versuchsfahrzeug genutzten Echtzeit Ethernet, als am geeignetsten herausgestellt. Es hat sich gezeigt, dass die Datenmenge, die über das Netzwerk übertragen werden muss, durch den Einsatz von ROIs bei der Kompression, reduziert werden kann. Es hat sich jedoch auch gezeigt, dass die benötigte Rechenzeit für ein Bild bei Einsatz von ROI Kompression in den meisten Fällen erhöht wird. Wegen der verwendeten Hardware für die Kameragateways, konnten bei dieser Arbeit keine hohen Frameraten erreicht werden. Aufgrund der Pipelinestruktur des Programms und des Aufbaus mit Threads, ist bei einem möglichen Austausch der Hardware, vor allem bei Mehrkernprozessoren, eine erhebliche Steigerung der Framerate zu erwarten. Bei dem jetzigen Stand des Programms kann eine rechteckige ROI zum Start des Programms festgelegt werden. In einer späteren Erweiterung wäre denkbar, die ROI während der Laufzeit anpassbar zu machen, um sich an Situationsänderungen während der Fahrt anzupassen. Es wäre außerdem denkbar, ROIs zu unterstützen, die nicht rechteckig sind. Dies wäre einfach umzusetzen, da hierfür nur eine einzelne Methode ausgetauscht werden muss.

Literaturverzeichnis

[CoRE Research Group] CoRE Research Group: *CoRE Research Group Website*. CoRE Research Group. – URL http://core.informatik.haw-hamburg.de/en/. – Zugriffsdatum: 2016-01

[Golner und Mikhael 2000] Golner, Mitchell A. ; Mikhael, Wasfy B.: Region Based Variable Quantization for JPEG Image Compression . In: *Image (Rochester, N.Y.)* (2000), S. 604–607. ISBN 0780364759

[Golston 2004] Golston, Jeremiah: Comparing media codecs for video content. In: *Proc. Embedded System Conference* (2004), S. 18. – URL http://www.frontporchdigital.com/Resource/Files/ComparingMediaCodecsforVideoContent.pdf

[IDS Imaging a] IDS Imaging: *uEye Camera Manual*. IDS Imaging Development Systems GmbH. – URL https://en.ids-imaging.com/manuals/uEye_SDK/EN/uEye_Manual/index.html. – Zugriffsdatum: 2016-01

[IDS Imaging b] IDS Imaging: *UI-1240ML-C Datasheet*. IDS Imaging Development Systems GmbH. – URL https://en.ids-imaging.com/IDS/datasheet_pdf.php?sku=AB00182. – Zugriffsdatum: 2016-01

[ISO 1993] ISO: *Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 2: Video*. International Organization for Standardization. August 1993. – URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=22411. – Zugriffsdatum: 2016-01

[ISO 1994] ISO: *Digital compression and coding of continuous-tone still images*. International Organization for Standardization. Februar 1994. – URL http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18902. – Zugriffsdatum: 2016-01

[ISO 2000] ISO: *JPEG 2000 image coding system – Part 1: Core coding system*. International Organization for Standardization. Dezember 2000. – URL http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=27687. – Zugriffsdatum: 2016-01

[ISO 2003] ISO: *Coding of audio-visual objects – Part 10: Advanced Video Coding*. International Organization for Standardization. Dezember 2003. – URL http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=37729. – Zugriffsdatum: 2016-01

[Konran Udo Gerber] Konran Udo Gerber: *Bitrate Viewer Website*. Konran Udo Gerber. – URL http://www.winhoros.de/docs/bitrate-viewer/index.html. – Zugriffsdatum: 2016-01

[Kontron AG] Kontron AG: *pITX-SP Datasheet*. Kontron AG. – URL http://www.kontron.com/downloads/datasheet/datasheet_pitx-sp.pdf?product=88730. – Zugriffsdatum: 2016-01

[libjpeg-turbo a] libjpeg-turbo: *libjpeg-turbo USAGE instructions*. libjpeg-turbo Project. – URL https://raw.githubusercontent.com/libjpeg-turbo/libjpeg-turbo/master/usage.txt. – Zugriffsdatum: 2016-01

[libjpeg-turbo b] libjpeg-turbo: *libjpeg-turbo Website*. libjpeg-turbo Project. – URL http://www.libjpeg-turbo.org/. – Zugriffsdatum: 2016-01

[Salomon und Motta 2010] Salomon, David ; Motta, Giovanni: Video Compression. In: *Handbook of Data Compression* (2010), S. 855–952. – URL http://link.springer.com/10.1007/978-1-84882-903-9{_}9. ISBN 9780071363242

[Steiner 2008] Steiner, Wilfried: *TTEthernet Specification*. TTTech Computertechnik AG. November 2008. – URL http://www.tttech.com

[Tanenbaum 2009] Tanenbaum, Andrew S.: *Moderne Betriebssysteme - 3., aktualisierte Auflage*. Pearson Studium, Oktober 2009. – ISBN 978-3-8273-7342-7

[The Qt Company] The Qt Company: *Qt-Framework*. The Qt Company. – URL http://www.qt.io/qt-framework/. – Zugriffsdatum: 2016-01

[Vaisey und Gersho 1992] Vaisey, J. ; Gersho, a.: Image compression with variable block size segmentation. In: *IEEE Transactions on Signal Processing* 40 (1992), Nr. 8, S. 2040–2060. – URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=150005. – ISSN 1053587X

[Valgrind Developers] Valgrind Developers: *Valgrind Tool Suite*. Valgrind Developers. – URL http://valgrind.org/. – Zugriffsdatum: 2016-01

[Wallace 1991] Wallace, Gregory K.: The JPEG still picture compression standard. In: *Communications of the ACM - Special issue on digital multimedia systems* Bd. 34, April 1991, S. 30–44

[Wang u. a. 2012] Wang, Shiqi ; Fu, Jingjing ; Lu, Yan ; Li, Shipeng ; Gao, Wen: Content-aware layered compound video compression. In: *2012 IEEE International Symposium on Circuits and Systems* (2012), S. 145–148. – URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6271519. ISBN 978-1-4673-0219-7

[Wikimedia] Wikimedia: *DCT HelligkeitsÃ¤nderung*. Wikimedia. – URL https://upload.wikimedia.org/wikipedia/commons/2/23/Dctjpeg.png. – Zugriffsdatum: 2016-01

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 11. Februar 2016 Jannik Schick