



Hochschule für Telekommunikation Leipzig  
University of Applied Sciences

# ENTWICKLUNG EINER (SMARTPHONE-)APP ZUR EINFACHEN KOMPRESSION VON BILDERN - DOKUMENTATION -

Studienmodul *ICT*  
der Hochschule für Telekommunikation  
Leipzig



## **Prüfungsvorleistung - Bilddatenkompression**

vorgelegt von

**Stefan Czogalla, Maik Lorenz, Jan Sutmöller, Stephan Kaden**

5. Februar 2016

**Dozent:** Prof. Dr. habil. Tilo Strutz

# INHALTSVERZEICHNIS

<b>Abbildungsverzeichnis</b>	<b>3</b>
<b>1 Theoretischer Teil</b>	<b>4</b>
1.1 Einleitung . . . . .	4
1.2 Was ist Bildkompression? . . . . .	4
1.3 Menschliches Sehen . . . . .	5
1.4 Digitalisierung . . . . .	5
1.4.1 Bilderzeugung . . . . .	5
1.4.2 Abtastung . . . . .	6
1.4.3 Quantisierung . . . . .	8
1.5 Kompressionsarten . . . . .	9
1.5.1 Verlustbehaftete Kompression . . . . .	9
1.5.2 Verlustfreie Kompression . . . . .	9
1.5.3 Vergleich Tabellarisch . . . . .	10
1.6 Skalare Quantisierung . . . . .	10
1.7 Vektorquantisierung . . . . .	10
1.8 Das RGB-Farbmodell . . . . .	10
1.9 skalare Farbquantisierung . . . . .	12
1.10 Zusammenfassung . . . . .	13
1.11 Literaturverzeichnis . . . . .	13
1.12 Definitionen, Akronyme, Abkürzungen . . . . .	14
<b>2 Vergleich der vorhandenen drei APPs</b>	<b>14</b>
2.1 Speicherplatzbedarf . . . . .	14
2.2 Speicherbelastung im Live-Betrieb . . . . .	15
2.3 Vergleich der vorhandenen drei APPs . . . . .	17
2.4 Beispielbilder . . . . .	20
2.5 Produktfunktionalität . . . . .	29
2.6 Randbedingungen . . . . .	29
<b>3 Anforderungen</b>	<b>29</b>
3.1 Fachkonzept . . . . .	29
3.2 Installationsprozedur . . . . .	29
3.3 Anforderung an die Entwicklung . . . . .	29
3.3.1 Entwicklungs-Umgebung . . . . .	29
3.3.2 Änderungsmanagement . . . . .	29
<b>4 Ergebnis</b>	<b>30</b>
4.1 Übernommene Funktionalitäten . . . . .	30
4.2 Nicht übernommene Funktionalitäten . . . . .	30
4.3 Eigene Entwicklungen . . . . .	31
4.4 Umfang der APP . . . . .	31
4.5 Erscheinungsbild . . . . .	32

4.6	Quellcode . . . . .	35
4.6.1	Allgemein . . . . .	35
4.6.2	MainActivity.java . . . . .	35
4.6.3	Pixel.java . . . . .	39
4.6.4	Skalar.java . . . . .	43
4.7	Beispielbilder . . . . .	45
<b>5</b>	<b>Anhang</b>	<b>55</b>
5.1	PVL Aufgabenbeschreibung . . . . .	56
5.2	Analyseergebnisse durch System Monitor . . . . .	62

## ABBILDUNGSVERZEICHNIS

1.1 Prinzip der Bilderzeugung . . . . .	5
1.2 Prinzip der Abtastung und Quantisierung . . . . .	6
1.3 Vergleich Bildraster . . . . .	7
1.4 Abtastung . . . . .	7
1.5 Alias-Effekt . . . . .	8
1.6 RGB-Farbmischung . . . . .	11
1.7 RGB-Farbwürfel(11) . . . . .	12
1.8 RGB-Farbbild mit 3 x8-Bit-Komponenten (8) . . . . .	13
2.1 Speicherplatzbedarf . . . . .	15
2.2 Speicherbelastung durch System Monitor . . . . .	16
2.3 Speicherbelastung durch Memory/- und CPU-Monitor . . . . .	16
2.4 Ergebnisse AndroidStudio Memory- und CPU-Monitor . . . . .	17
2.5 FotoQuant Original . . . . .	21
2.6 FotoQuant Lloyd . . . . .	22
2.7 FotoQuant MidTread . . . . .	23
2.8 QuanPic Original . . . . .	24
2.9 QuanPic MedianBlur . . . . .	25
2.10 QuantiPic Originalbild . . . . .	26
2.11 QuantiPic Helligkeit . . . . .	27
2.12 QuantiPic Farbwerte . . . . .	28
4.1 QuantiPig Logo . . . . .	31
4.2 QuantiPig Startbildschirm . . . . .	32
4.3 QuantiPig-Menü Verarbeitungsverfahren . . . . .	33
4.4 QuantiPig-Menü Quantisierungsintervall . . . . .	34
4.5 signedunsigned . . . . .	42
4.6 QuantiPig Originalbild . . . . .	46
4.7 QuantiPig Pixel Modus Intervall 1 . . . . .	47
4.8 QuantiPig Pixel Modus Intervall 2 . . . . .	48
4.9 QuantiPig Pixel Modus Intervall 4 . . . . .	49
4.10 QuantiPig Pixel Modus Intervall 8 . . . . .	50
4.11 QuantiPig Skalar Modus Intervall 1 . . . . .	51
4.12 QuantiPig Skalar Modus Intervall 2 . . . . .	52
4.13 QuantiPig Skalar Modus Intervall 4 . . . . .	53
4.14 QuantiPig Skalar Modus Intervall 8 . . . . .	54

# 1 THEORETISCHER TEIL

## 1.1 EINLEITUNG

Mit dem Zeitpunkt der Erfindung des Computers ist eine Aufgabe der Informatik eine Möglichkeit für die Kompression und Kodierung von Daten zu finden. Spätestens mit der Erfindung von grafischen Ausgabegeräten wie Monitor und Drucker ist der Bedarf an einer effizienten und platzsparenden Speicherung, sowie Übermittlung, von Daten stetig gewachsen. In nahezu allen technischen Bereichen werden heutzutage digitale Bilder verwendet. Neben der digitalen Fotografie, der Archivierung von Bilddokumenten und der technischen Qualitätskontrolle kommen digitale Bilder zum Großteil auf Webseiten zum Einsatz. Würden hier dieselben Qualitätsansprüche an digitale Bilder gestellt werden, wie man sie an die klassische Fotografie stellt, würden enorme Datenmengen von mehreren Megabyte pro Bild entstehen. Deshalb ist eine effiziente und platzsparende Kompression unerlässlich. Diese Arbeit geht kurz auf die Definitionen und Grundlagen der Bildkompression ein und veranschaulicht dies durch die Quantisierung in drei verschiedenen Applikationen. Eine dieser drei Applikationen wird im Laufe dieser Arbeit mit den gewonnenen Erkenntnissen verbessert und vereint die positiven Eigenschaften der anderen Applikationen. (1)

## 1.2 WAS IST BILDKOMPRESSION?

Doch zunächst klären wir was unter Bildkompression verstanden wird. Wie jede Datenkompression, beruht auch die Bildkompression darauf aus dem ursprünglichen Datensatz Daten zu entfernen, deren Verlust kaum wahrnehmbar ist oder welche vollständig rekonstruierbar sind. Datenkompression ist ein Verfahren zur Reduktion des Speicherbedarfs von Daten, zu einem solchen Verfahren gehören logischerweise Methoden zur Reduktion des benötigten Speicherbedarfs (Kompression) und zur Wiederherstellung dieser Daten in die ursprüngliche Form (Dekompression). Nach jeder Kompression gilt immer folgende Gleichung:

$$\text{komprimierte Datei} = \text{Originaldatei} - \text{Redundanz}$$

„Unnötige“ Daten kann man im allgemeinen als Redundanz bezeichnen, also Daten, die für die eigentliche Information nicht wichtig sind, da sie entweder für den Menschen nicht wahrnehmbar sind oder sie komprimiert werden können (z.B. wiederkehrende Muster). Das bedeutet im Umkehrschluss, dass Daten die keine Redundanzen enthalten nicht komprimiert werden können. Dasselbe gilt natürlich für Daten deren Redundanzen nicht erkannt werden. Wird die unkomprimierte Datenmenge durch einen Wert dividiert, so erhält man die komprimierte Datenmenge. Dieser Wert wird als Kompressionsrate bezeichnet. „Üblich ist die Angabe als Quotient in der Form x : 1. Es ergibt sich also die Gleichung:

$$\text{Datenmenge(kompr.)} = \text{Datenmenge(unkompr.)} : \text{Kompressionsrate}$$

Ein Beispiel: Die unkomprimierte Datenmenge beträgt 300 MB und die Kompressionsrate beträgt 20 : 1. Somit ergibt sich für die Datenmenge(kompr.) =  $300 \text{ MiB} : 20 : 1 = 300 \text{ MiB} : 20 = 15 \text{ MiB}$  (2)

### 1.3 MENSCHLICHES SEHEN

Welcher Zusammenhang besteht nun zwischen Redundanz, also „unnötigen“ Daten und dem menschlichen Auge. Im Auge bildet die Linse das Originalbild auf Netzhaut (Retina) ab. Diese enthält Fotorezeptoren, welche aus rund 130.000.000 und 7.000.000 Zäpfchen besteht. Die Stäbchen registrieren Intensität (Hell-Dunkel-Verhältnis), wohingegen die Zäpfchen für die Farbregistrierung zuständig sind, je ein Zäpfchentyp für Grün, Rot, Blau. Durch Mischung der drei RGB-Farbkanäle lässt sich redundanzarm ein visueller Eindruck erzeugen, der dem natürlich wahrgenommenen Bild recht ähnlich ist. Das menschliche Auge kann circa 60 verschiedene Grautöne unterscheiden, daher reichen  $2^8 = 256$  Grauwerte aus um einen visuellen Eindruck zu erreichen.

### 1.4 DIGITALISIERUNG

Um analoge Bilder in digitalen Systemen nutzen zu können müssen diese zunächst in eine geeignete numerische Darstellung gebracht werden, dazu sind der Definitionsbereich, d.h. die Menge der erlaubten Punkte (Pixel) der Wertebereich (Grau- / Farbwerte) in endlich viele Intervalle aufzuteilen. Dieser Gesamtorgang wird als Digitalisierung bezeichnet. Die Digitalisierung des Definitionsbereiches wird als Rasterung (Scanning) und die Digitalisierung des Wertebereiches wird als Quantisierung (Sampling) bezeichnet.

#### 1.4.1 BILDERZEUGUNG

Prinzipiell wird ein Objekt bei der Bilderzeugung über eine Linse auf der Bildebene abgebildet (siehe Abbildung: 1.1). Hier setzt die Digitalisierung ein. In der Realität können nur Bilder endlicher Ausdehnung behandelt werden, daher muss das Bild oder besser der Bildausschnitt gefenstert werden (Fensterung). Außerdem können nur endlich viele Signale verarbeitet werden, daher wird die Projektion des Objektes auf der Bildebene abgetastet (Abtastung/Sampling). Die einzelnen Intensitätswerte werden in einer Bildmatrix übernommen.

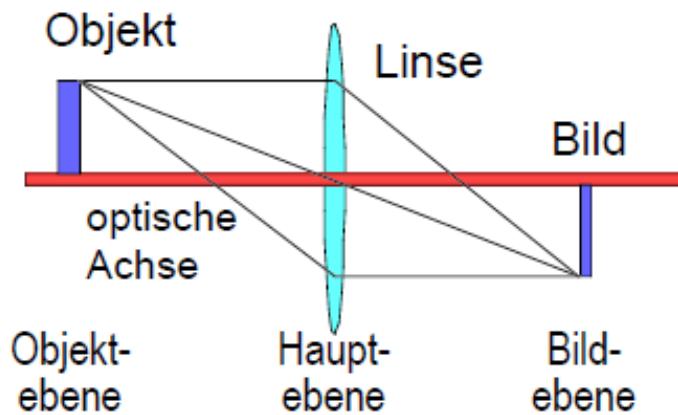


Abbildung 1.1: Prinzip der Bilderzeugung

Es können des Weiteren nur endlich viele Signalamplituden verarbeitet werden (Quantisierung), daher werden den einzelnen Bildpunkten (Pixeln) konkrete Werte zugeordnet. Wie in Abschnitt 3 werden die Intensitätswerte durch Graustufen dargestellt, somit werden den Pixeln also konkrete Graustufen zugeordnet (siehe Abbildung: 1.2). Dasselbe Prinzip gilt auch für die erwähnten Farbwerte.

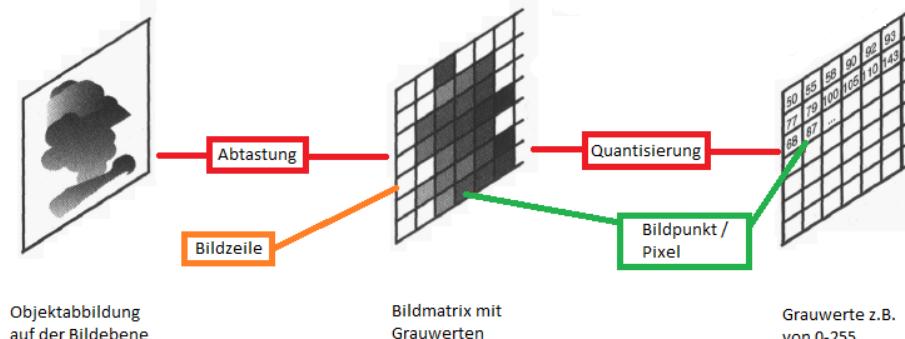


Abbildung 1.2: Prinzip der Abtastung und Quantisierung

#### 1.4.2 ABTASTUNG

Die Projektionsfläche auf der Bildebene wird durch ein regelmäßiges Raster von Photosensoren abgetastet. Je nach Dichte der Abtastpunkte pro Flächeneinheit entsteht ein hochauflösendes oder ein "grobkörniges" Bild. Unter Abtastung versteht man im Allgemeinen „die Erfassung eines zeitkontinuierlichen analogen Signals in bestimmten Zeitabständen. Zu diesem Zweck wird das Analogsignal zu den Abtastzeitpunkten t<sub>1</sub>, t<sub>2</sub> usw. abgetastet und der dabei erhaltene Signalwert einer Halteschaltung zugeführt. Die Abtastung erfolgt in aller Regel gemeinsam mit der Spannungshaltung in einer Abtast- und Halteschaltung, Sample and Hold. Die Zeitabstände zwischen den einzelnen Abtastungen ist die Abtastrate.“ (3) Bei biologischen Augen sind die Photorezeptoren in einer "Wabenstruktur" angeordnet. Auch bei neueren technischen Entwicklungen (elektronische Chip-Augen) werden hexagonale Gitter benutzt. Vorteile sind hier die bessere Chip-Flächenfüllung bei integrierter Signalverarbeitungselektronik, sowie die homogenen Pixelabstände. Andere technische bildverarbeitende Systeme benutzen (heute) fast ausschließlich ein kartesisches Basisgitter für das Bildraster, welches aus rechteckigen bzw. quadratischen Pixeln besteht. Der Vorteil eines kartesischen Rasters liegt in der einfachen mathematischen Behandlung als Matrix, jedoch sind in diesem Raster nicht alle Pixelabstände homogen. Sie sind also inhomogen (siehe Abbildung: 1.3). Die Größe der Gittermaschen, die über das Urbild gelegt werden, hat einen entscheidenden Einfluss auf die Qualität des digitalisierten Bildes. Bei der Wahl eines zu groben Rasters gehen Details verloren oder feine Bilddetails werden verzerrt wiedergegeben. Bei einem zu feinen Gitter steigt die Auflösung, aber auch die Dateigröße. Da eine zu grobe Rasterung zu einer Verschmelzung der Objektdetails führen kann gilt folgendes: „Die Rasterweite eines Bildes darf nicht größer sein als die halbe geometrische Ausdehnung des kleinsten noch abzubildenden Objektdetails.“ (3) Ein ähnliches Prinzip gilt für die Abtastung. Hier

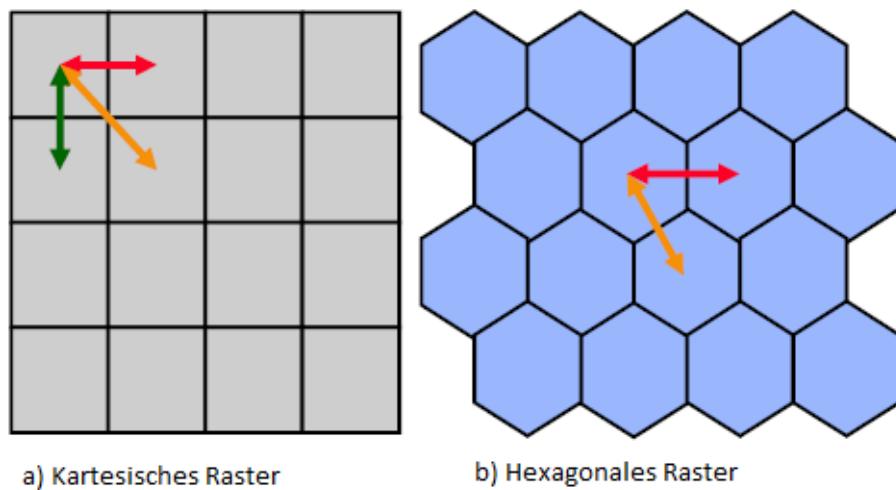


Abbildung 1.3: Vergleich Bildraster

greift das Nyquist-Shannonsche Abtasttheorem, welches besagt, dass ein kontinuierliches Signal (mit einer Maximalfrequenz  $f_{\max}$ ) mit einer Frequenz größer als  $2 * f_{\max}$  abgetastet werden muss, damit aus dem so erhaltenen zeitdiskreten Signal das Ursprungssignal ohne Informationsverlust rekonstruiert werden kann.

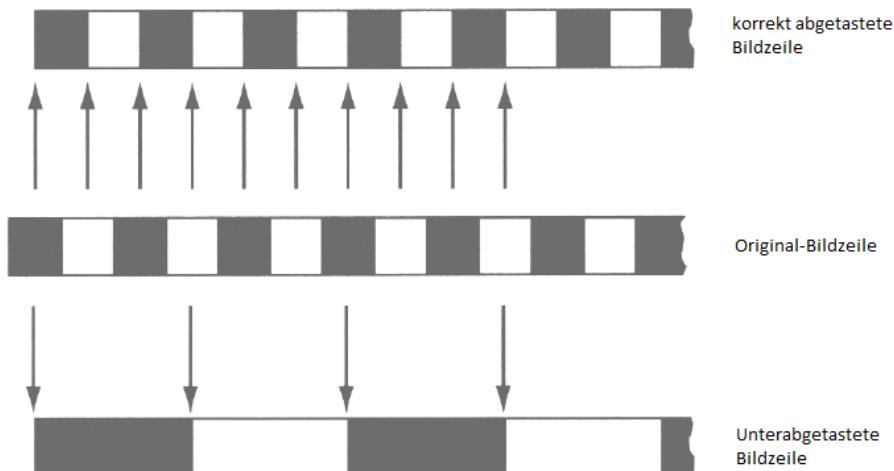


Abbildung 1.4: Abtastung

In Abbildung: 1.4 erkennt man, dass die Abtastrate doppelt so groß wie die Detailfrequenz ist und somit das Original wiedergegeben werden kann. Bei zu geringer Abtastrate wird das Original verfälscht. Über diesen Effekt, der Unterabtastung, muss man sich klar machen, dass je nach Abtastrate auch noch viel drastischere Effekte eintreten können: Wenn der Abtastpunkt etwa alle zwei Kästchen kommt (also Detailfrequenz = Abtastfrequenz) und immer im "Schwarzen" des Originals liegt, dann wird der Sample& Hold - Wert immer

schwarz sein (siehe Abbildung: 1.5 Alias-Effekt)! Dies ist der Effekt des Aliasing: Frequenzen, die für die Abtastrate zu hoch sind, sehen so aus wie tiefe. Die hohen Frequenzen geben sich sozusagen als jemand anderes aus, daher die Bezeichnung Alias.

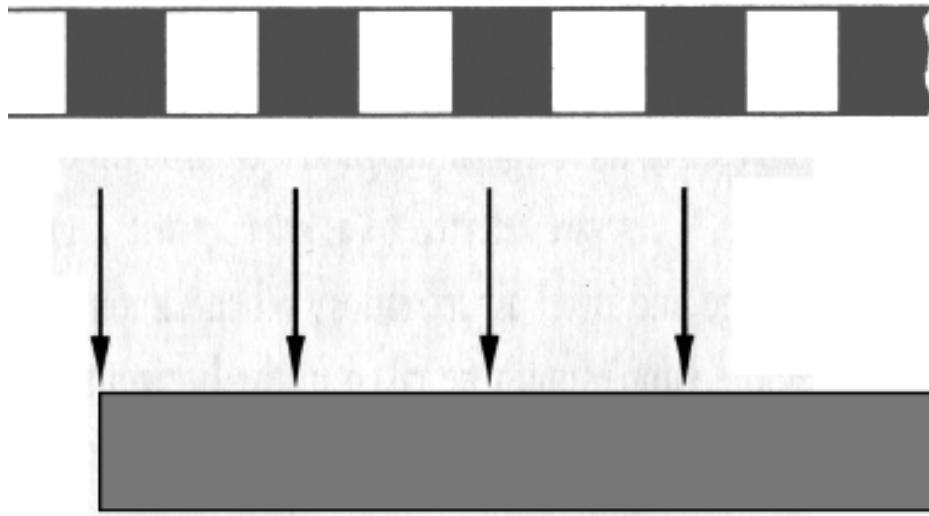


Abbildung 1.5: Alias-Effekt

#### 1.4.3 QUANTISIERUNG

Quantisierung ist, wie in den vorherigen Abschnitten beschrieben Teil der Digitalisierung von analogen Signalen. Mit dem Begriff Quantisierung wird die Zuordnung der digitalisierten Grauwerte zu den Pixeln bezeichnet. Als Werte werden meist die natürlichen Zahlen von 0 bis 255 verwendet. Neben der Rasterfeinheit wird die Bildqualität durch die Anzahl der Graustufen zwischen Schwarz und Weiß bestimmt. Eine Reduzierung der Graustufen eines Grautonbildes führt zu größeren homogenen Flächen mit störenden Kanten. Die meisten Bildverarbeitungssysteme besitzen 256 oder sogar 1024 Graustufen. Man sagt auch, sie können mit 8 Bit bzw. 10 Bit quantisieren ( $2^8 = 256$ ;  $2^{10} = 1024$ ). Die Digitalisierung von Farbbildern entspricht der von Grautonbildern mit dem einzigen Unterschied, dass das Farbbild in seine Rot-, Grün- und Blauanteile durch Filter zerlegt wird. Für jeden Farbanteil gibt es eine eigene Bildmatrix. Die Rot-, Grün- und Blaumatrizen lassen sich zum Originalbild kombinieren. Da für jede der drei Matrizen 256 bzw. 1024 Grauwerte für die Quantisierung zur Verfügung stehen, beträgt die Zahl der theoretisch realisierbaren Farbwerte 16,7 Millionen (24bit) bzw. 1,07 Milliarden (30bit). Bilder stellen große Dateien dar, die in bestimmter Weise organisiert sind. Dabei wird besonders viel Wert auf die Datenreduktion gelegt, ohne die Qualität der Bilder wesentlich oder überhaupt nicht zu mindern. Neben der Datenreduktion durch die Momentanwertabtastung, bei der die Datenreduktion nur durch Verlust von Daten auftritt. Eine sehr einfache Methode zur Datenreduktion demnach die Quantisierung, bei der die Genauigkeit reduziert, wird mit der die Daten abgespeichert werden.

## 1.5 KOMPRESSSIONSARTEN

Bei den Kompressionsverfahren unterscheidet man zwischen drei verschiedenen Arten:

- Signalkompression: Die Reduzierung der Redundanz im Signal wird hierbei durch die Betrachtung jedes einzelnen Pixels, unabhängig von allen anderen Pixeln erreicht.
- Umgebungsbasierter Kompression: Die Redundanz zwischen benachbarten Pixeln wird reduziert, indem eine Pixelfolge betrachtet wird.
- Wahrnehmungsorientierte Kompression: Die Reduktion der Information, die für die Wahrnehmung der Bilddateien relevant ist. Zum Beispiel die Reduktion des Wertebereichs auf 64 Helligkeitswerte mit der Begründung, dass rund 60 Werte durch den Betrachter unterschieden werden können. (4)

Verfahren, die einer der ersten beiden Strategien folgen, führen zu einer Verlustfreie Kompression. Durch die Reduktion von Informationen (Wahrnehmungsorientierte-kompression) können höheren Kompressionsraten erreicht werden. Aufgrund der sehr hohen Kompression gehen hier Daten verloren und die Rekonstruktion ist nur näherungsweise möglich, daher spricht man von einer verlustbehaftete Kompression.

### 1.5.1 VERLUSTBEHAFTETE KOMPRESION

Diese Kompressionsart wird verwendet, wenn Bildinformation übertragen werden müssen, bei der Details nicht den Informationsgehalt des Bildes bestimmen. Hier findet eine Reduktion der Bilddaten statt, sodass das Ursprungsbild nicht 1:1 wiederherstellbar ist. Die Fehler, die bei zu starker Datenreduktion sichtbar werden, nennt man Artefakte. Einsatzgebiete sind z.B. digitales Fernsehen, Telekonferenzen, Bilder im Internet. Bei Netzwerken wie dem Internet kommt noch hinzu, dass die Übertragungsgeschwindigkeiten meist sehr niedrig sind, und da verlustbehaftete Kompression vor allem bei True-Color Bildern eine viel höheren Kompressionsfaktor erreicht als die Verlustfreie, wählt man hier diese Kompressionsart. Hier gilt der Grundsatz „Speed before Quality“. Formate die verlustbehaftete Kompression unterstützen sind z.B.: GIF, JPEG, Wavelet.

### 1.5.2 VERLUSTFREIE KOMPRESION

Wie vom Namen her zu raten ist, soll bei diesem Verfahren keine Information verloren gehen. Die verlustfreie Kompression kommt vorwiegend in der professionellen Bildbearbeitung zum Einsatz, wo mit teureren, schwer zu beschaffenden oder aufwendig zu berechnenden Bilddaten gearbeitet wird. Hier ist jede Bildinformation wichtig. Es findet also keine Reduktion der Daten statt. Im Gegensatz zur verlustbehafteten Kompression gilt hier „Quality before Speed“. Einsatzgebiete sind z.B. Satellitenbilder, medizinische oder künstliche Bilder. Formate die verlustfreie Kompression unterstützen sind z.B.: BMP, GIF, PNG, JPEG, TIF. Das Prinzip beruht darauf, die Daten anders als vorher zu organisieren, indem Wiederholung von Strukturen erkannt und hierarchisch dargestellt werden. Zum Beispiel wird eine sich wiederholende Bitfolge einmal in einer Art Wörterbuch abgelegt und dann nur noch durch ihre Nummer repräsentiert.

### 1.5.3 VERGLEICH TABELLARISCH

Die wichtigsten Unterschiede zwischen den beiden Verfahren sind aus dem Qualitätsunterschied eines Bildes, vor einer Kompression und nach der wieder Dekomprimierung des gleichen Bildes, sichtbar (siehe Abbildung 1.1 Tabellarischer Vergleich).

Verlustfreie	Verlustbehaftete
<ul style="list-style-type: none"> <li>+ keine Zerstörung der Bilddaten</li> <li>+ Dekompression rekonstruiert</li> <li>+ Ursprungsdaten vollständig</li> <li>- niedrige Kompressionsrate erreichbar</li> <li>- hoher Übertragungsaufwand</li> </ul>	<ul style="list-style-type: none"> <li>+ hohe Kompressionsrate erreichbar</li> <li>- Verringerung der Informationsdichte</li> <li>- Dekompression rekonstruiert</li> <li>- Ursprungsdaten nicht vollständig</li> <li>+ niedriger Übertragungsaufwand</li> </ul>

Tabelle 1.1: Tabellarischer Vergleich

## 1.6 SKALARE QUANTISIERUNG

Die skalare Quantisierung ordnet jedem Signalwert einen quantisierten Wert aus einer endlichen Wertemenge zu. Die Zuordnung erfolgt dabei im einfachsten Fall linear auf Basis eines Rasters mit Intervallen fester Länge. Alle Samples innerhalb eines bestimmten Intervalls werden dabei auf denselben quantisierten Wert abgebildet, wodurch verlustbehaftete Datenkompression entsteht. Anstelle eines festen Rasters können auch unterschiedliche Intervallbreiten gewählt werden, um bestimmte Werte stärker zu quantisieren als andere. Dies kann beispielsweise Sinn machen, wenn sich dadurch Einschränkungen der menschlichen Wahrnehmung ausnutzen lassen und wird als nichtlineare Quantisierung bezeichnet.

## 1.7 VEKTORQUANTISIERUNG

Die Vektorquantisierung berücksichtigt  $n$  Signalwerte gleichzeitig, die als Vektor des  $n$ -dimensionalen Raums aufgefasst werden. Wie die skalare Quantisierung stellt auch die Vektorquantisierung einen verlustbehafteten Vorgang dar. Ein Vektorquantisierer der Dimension  $n$  und Größe  $s$  bildet Eingabevektoren auf eine endliche Menge  $C$  ab, die aus  $s$  Ausgabevektoren besteht. Für die Wahl der Ausgabevektoren aus  $C$  können verschiedene Kriterien herangezogen werden. Im einfachsten Fall kommt das euklidische Abstandsmaß der Vektoren zum Einsatz. Die Menge  $C$  der Ausgabevektoren wird als Codebuch bezeichnet. Die größte Herausforderung bei der Vektorquantisierung ist die Wahl eines geeigneten Codebuchs. Dieses muss in einer Trainingsphase mit Hilfe charakteristischer Signalvektoren optimiert und so an typische Signalstatistiken angepasst werden. Ein verbreiteter Algorithmus zur Codebuch-Erstellung ist der LBG-Algorithmus.

## 1.8 DAS RGB-FARBMODELL

Um Farben quantitativ beschreiben zu können, gibt es sogenannte Farbmodelle. Da für die Beschreibung des folgenden Verfahrens zur Farbquantisierung vorwiegend das RGB-

Farbmodell genutzt wird, soll es hier genauer erläutert werden. Alle Farbstufungen entstehen beim RGB-Farbmodell durch Mischen der drei Grundfarben Rot, Grün, Blau (RGB). Durch das beschränkte räumliche Auflösungsvermögen unseres Auges werden die drei Farbkomponenten als Überlagerung (additive Farbmischung), also als einheitlicher Farbreiz, wahrgenommen.(s. Abbildung 1.6)

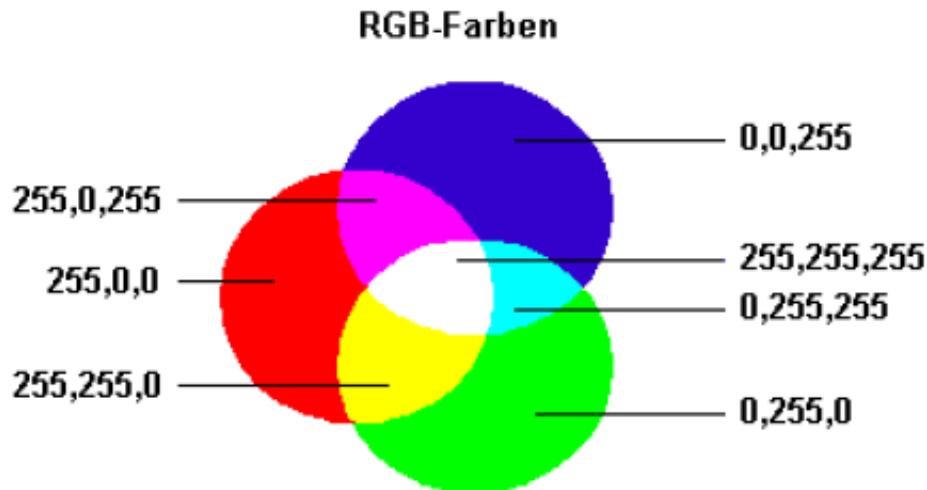


Abbildung 1.6: RGB-Farbmischung

Im RGB-Farbmodell wird jede Farbe also durch ein Tripel aus den Grundfarben Rot, Grün und Blau beschrieben. Aus diesem Grund kann das RGB-Farbmodell als Farbwürfel dargestellt werden. Diesem Würfel liegt ein kartesisches Koordinatensystem zugrunde mit den 3 Achsen R; G; B. Innerhalb des Farbwürfels und an den begrenzenden Flächen ergeben sich alle erzeugbaren Farben. Diese Anzahl der Farben ist abhängig von den Intensitäten entlang der Achsen. Ist die Einstellung der Stärke der Beleuchtung in 256 Einheiten geteilt (numerisch also durch 1 Byte unterscheidbar), ergeben sich  $2^8 * 2^8 * 2^8 = 16777216$  mögliche Farbstufungen (entspricht einer Farbtiefe von 24 Byte).(s. Abbildung 1.7)

Farben werden im RGB-Modell also als dreidimensionalen Vektorraum definiert. Die Vektoren dieses Raumes heißen Farbvalenzen, die Länge eines Vektors ist ein Maß für die Leuchtdichte und heißt Farbwert, seine Richtung bestimmt die Farbart. Die Basisvektoren heißen Primärvalenzen. Mit den Primärvalenzen R; G; B lässt sich also für jede Farbvalenz F eine Farbgleichung aufstellen:

$$F = r * R + g * G + b * B$$

Es sei hier erwähnt, dass es neben dem RGB-Farbmodell noch andere Modelle gibt. Diese werden hier nur genannt, da sie für die weitere Betrachtung keine Rolle spielen. Weitere Farbmodelle sind das CMY(K) - Farbmodell, HSV - Farbmodell, YUV - Farbmodell und das CIE XYZ – Farbmodell.

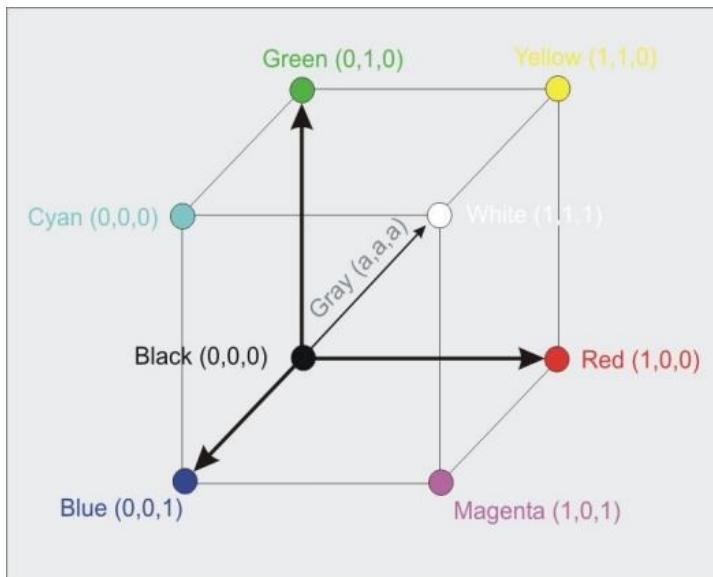


Abbildung 1.7: RGB-Farbwürfel(11)

## 1.9 SKALARE FARBQUANTISIERUNG

Bei skalaren Farbquantisierung wird jede der ursprünglichen Farbkomponenten C im Wertebereich [0...m-1] unabhängig von einander in den neuen Wertebereich [0...n-1] überführt, im einfachsten Fall durch eine lineare Quantisierung in der Form

$$C' \leftarrow \lceil C * \frac{n}{m} \rceil$$

für alle Farbkomponenten C. Ein Beispiel ist die Konvertierung eines Farbbilds mit 3 x 12-Bit-Komponenten, also 12 Bit pro Farbkanal mit m = 4096 möglichen Werten in ein herkömmliches RGB-Farbbild mit 3 x 8-Bit-Komponenten, also jeweils n = 256 Werten. Jeder Komponentenwert wird daher durch  $4096/256 = 16 = 2^4$  ganzzahlig dividiert oder, anders ausgedrückt, die untersten 4 Bits der zugehörigen Binärzahl werden einfach ignoriert. Abbildung 9 zeigt die skalare Quantisierung von Farbkomponenten durch Abtrennen niedrigerwertiger Bits. Quantisierung von 3 x 12-Bit- auf 3 x 8-Bit-Farben. (s. Abbildung 1.8 RGB-Farbbild)

Die skalare Quantisierung nimmt keine Rücksicht auf die Verteilung der Farben im ursprünglichen Bild. Sie wäre ideal, wenn die Farben im RGB-Würfel gleichverteilt sind. Bei natürlichen Bildern ist die Farverteilung in der Regel jedoch ungleichförmig, sodass einzelne Regionen des Farbraums dicht besetzt sind, während andere Farben im Bild überhaupt nicht vorkommen. Der durch die skalare Quantisierung erzeugte Farbraum kann zwar auch die nicht vorhandenen Farben repräsentieren, dafür aber die Farben in dichten Bereichen nicht fein genug abstuften.

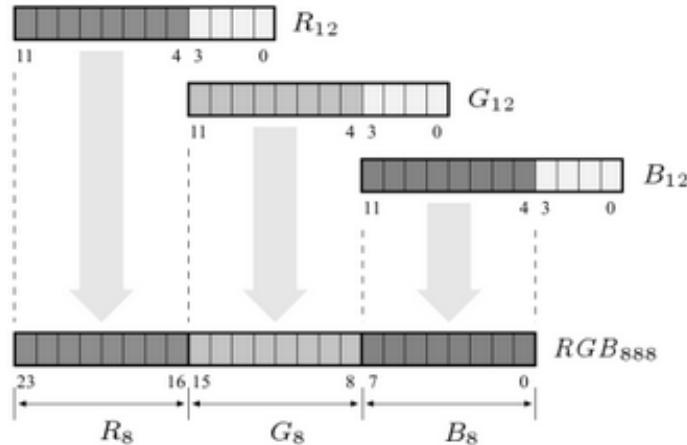


Abbildung 1.8: RGB-Farbbild mit 3 x 8-Bit-Komponenten (8)

## 1.10 ZUSAMMENFASSUNG

Die Digitalisierung besteht aus Abtastung und Quantisierung. Durch die beschriebenen Eigenschaften der Abtastung und Quantisierung lässt sich also festhalten, dass ein digitales Bild immer nur eine Annäherung der Originalabbildung ist. Wir haben herausgestellt, die Abtastung ist die Aufteilung des Bildes in Bildpunkte (Pixel) und die Quantisierung die Bewertung der Helligkeit (Intensität) eines Pixels mittels einer festgelegten Grauwert- bzw. Farben-Menge, z.B. natürliche Zahlen von 0 bis 255. Die Qualität der Datenkompression eines digitalen Bildes hängt also maßgeblich von diesen beiden Faktoren ab. Wie wir herausgestellt haben ist die skalare Farbquantisierung ein einfaches und schnelles Verfahren, das den Bildinhalt selbst nicht berücksichtigt, zur Datenreduktion. Durch den zeitlich begrenzten Rahmen und Umfang dieser Arbeit in Hinsicht auf die Programmierung haben wir uns für die skalare Farbquantisierung entschieden, da die Vektorquantisierung maßgeblich von der Programmierung eines optimalen Codebuches abhängt. Ein zweiter Punkt für die skalare Quantisierung ist, dass in dieser Arbeit das Grundlegende Verständnis einer Quantisierung und deren Funktionsweise dargestellt werden soll.

## 1.11 LITERATURVERZEICHNIS

1. **Francis, Chukwumezie Millverton.** Vortragsskript der TU München. Proseminar : Grundlagen Bildverarbeitung/Bildverständen Bildkompression. [PDF]. 21. 12 2005.
2. **www.mathemedien.de.** [Online] [Zitat vom: 28. 01 2016.] <http://www.mathemedien.de/datenkompression.html>
3. **Neumann B.** Bildverarbeitung für Einsteiger: Programmbeispiele mit Mathcad. s.l. : Springer Verlag, 2005.
4. **Tönnies, K.** Grundlagen der Bildverarbeitung. s.l. : Pearson Studium, 2005.

5. **Strutz, T.** Bilddatenkompression. s.l. : Vieweg+Teubner Verlag, 2005.
6. **www.itwissen.info** [Online] [Zitat vom: 29. 01 2016.] <http://www.itwissen.info/definition/lexikon/Quantisierungsfehler-quatization-error.html>.
7. **www.itwissen.info** [Online] [Zitat vom: 29. 01 2016.] <http://www.itwissen.info/definition/lexikon/Abtastung-sampling.html>.
8. **Burger, W. & Burge** Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java. s.l. : Springer Verlag, 2006.
9. **Ohm, J.** Digitale Bildcodierung: Repräsentation, Kompression und Übertragung von Bildsignalen . s.l. : Springer Verlag, 2013.
10. **Schmitz, R.** et al. Kompendium Medieninformatik: Mediennetze. s.l. : Springer Verlag, 2006.
11. **homepages.thm.** [Online] [Zitat vom: 01.02.2016.]  
[https://homepages.thm.de/~hg10013/Lehre/MMS/SS01\\_WS0102/Farbmodelle/Kapitel/Kapitel4.html](https://homepages.thm.de/~hg10013/Lehre/MMS/SS01_WS0102/Farbmodelle/Kapitel/Kapitel4.html)

## 1.12 DEFINITIONEN, AKRONYME, ABKÜRZUNGEN

**APP** Kurzform für Applikation

**XML** Extensible Markup Language

**GUI** Graphical User Interface

**QuantiPig** quantisiertes Picture

## 2 VERGLEICH DER VORHANDENEN DREI APPS

Als Ausgangssituation wurden der Studentengruppe drei APPs aus vorherigen Matrikeln vorgelegt, die ebenfalls Bilddaten komprimieren. Diese drei APPs gilt es zu vergleichen und die Stärken herauszuarbeiten. Anhand dieser Ergebnisse gilt es eine neue APP zu entwickeln.

### 2.1 SPEICHERPLATZBEDARF

Die Speicherbelastung wurde anhand folgender Aspekte betrachtet:

- benötigter Speicherplatz der APP auf dem Smartphone
- Speicherbelastung während des Betriebs

	FotoQuant	QuanPic	QuantiPic	QuantiPig
Speicherbedarf:	4,72 MB	46,5 MB	7,28 MB	46,46 MB

Abbildung 2.1: Speicherplatzbedarf

Während der Analyse wurde ein HTC One (M7), mit dem bereits die Testfotos gemacht wurden, verwendet. Die Ergebnisse werden in tabellarischer Form dargestellt und mit einer Einschätzung am Ende zusammengefasst.

### Einschätzung

FotoQuant hat trotz seiner vielen Funktionen einen sehr geringen Speicherbedarf auf dem Smartphone. Nicht mal 5 MB werden in Anspruch genommen. Der Quellcode von 612 Zeilen Länge (FotoQuantCameraView.java; FotoQuantCameraListener.java; FotoQuant.java) wird nur bedingt dafür verantwortlich sein. Auch QuantiPic belegt mit seinen drei Quantisierungsverfahren nur knappe 7,3 MB Speicher auf dem Testgerät. Der Quellcode ist hier mit 267 Zeilen wesentlich kompakter (MainActivity.java; + import von OpenCV Klassen), das zusätzliche Integrieren von einigen OpenCV Klassen erhöht dabei jedoch die Gesamtgröße. Beide APP's integrieren OpenCV nicht von vorn herein, es muss separat herunter geladen und installiert werden.

Deutlich mehr Speicher belegt QuanPic. Mit fast 47 MB ist die Anwendungssoftware knapp zehn Mal größer als FotoQuant und das obwohl weniger Funktionen zur Verfügung stehen! Der Quellcode unterscheidet sich mit 696 Zeilen nur gering von den beiden Ersten APP's (MainActivity.java; NeueQuant.java). Der hohe Speicherplatzbedarf begründet sich allein durch das komplette Einbinden von OpenCV.

Aus den bestehenden APP's wurde QuanPic als Basis für QuantiPig gewählt. Erweitert um einige Vorteile aus FotoQuant und QuantiPic, teilt sich der Quellcode nun auf drei Hauptdateien. Mit 456 Zeilen (MainActivity.java; Pixel.java; Skalar.java) hat dieser die zweitkürzeste Länge. Kompakt und Funktional werden knapp 46,5 MB auf dem Testgerät belegt, denn auch hier ist OpenCV komplett integriert, so dass es keiner zusätzlichen Installation bedarf.

## 2.2 SPEICHERBELASTUNG IM LIVE-BETRIEB

Die Analyse der Speicherbelastung erfolgte durch die Auswertung von Speicher- und CPU-Auslastung während des Live-Betriebs. Dabei wurde unter anderem die APP „System Monitor“ auf dem Testgerät installiert und die Speicherbelastung aufgezeichnet. Als Testumgebung wurde vor jeder Aufzeichnung sichergestellt, dass nur die zu testende App sowie „System Monitor“ aktiv geöffnet waren. Während der Aufzeichnung wurden die einzelnen Modi der APP's durchgetestet. Eine Aufzeichnung war nicht länger als 3 Minuten, wobei alle 1000 ms der Speicherbedarf durch „System Monitor“ ausgelesen wurde. Die erfassten Werte wurden im Mittelwert zusammengefasst und werden nun tabellarisch aufgelistet. Die Rohdaten befinden sich im Anhang „Analyseergebnisse durch System Monitor“. (siehe 5.2) Eine zweite Möglichkeit die Speicherbelastung zu ermitteln erfolgte durch den „Memory

	FotoQuant	QuanPic	QuantiPic	QuantiPig
Speicherbelastung:	94.922,0625 kB	46.299,0588 kB	45.297,2414 kB	36.658,0377 kB

Abbildung 2.2: Speicherbelastung

Monitor“ und „CPU Monitor“ von Android Studio. Zur Vorbereitung wurde die entsprechende APP in Android Studio, sowie auf dem Testgerät gestartet. Das Monitoring ermöglicht eine grafische Auswertung der folgenden Punkte:

- Total Memory, allocted Memory, Free Memory
- Total CPU Usage, User Usage, Kernel Usage

Für die Analyse wurde die Belastung jeweils zum APP-Start, während der Modi sowie zum Beenden betrachtet. Die ermittelten Werte wurden wieder im Mittelwert zusammengefasst (s. Abbildung 2.3)

	FotoQuant	QuanPic	QuantiPic	QuantiPig
<b>Memory (MB):</b>	<b>59,12</b>	<b>11,65</b>	<b>12,79</b>	<b>25,73</b>
Free:	14,01	4,21	2,77	9,80
Allocated:	45,11	7,44	10,02	15,93
<b>CPU Usage (%):</b>	<b>41,95</b>	<b>30,05</b>	<b>27,74</b>	<b>35,24</b>
User:	29,50	26,64	24,88	30,83
Kernel:	12,45	3,41	2,87	4,41

Abbildung 2.3: Speicherbelastung durch Memory/- und CPU-Monitor

## Einschätzung

Beide Vorgehen machen deutlich, dass zwei APP's eine sehr geringe Speicherbelastung haben, während zwei APP's einen wesentlich höheren Anspruch besitzen. FotoQuant kann seinen Vorteil aus dem geringen Speicherplatzbedarf nicht durch eine geringe Speicherbelastung erweitern. Im Gegenteil, die APP ist in der Performance durch den fast dauerhaften Anspruch auf 45 MB Speicher und knapp 42% CPU-Leistung sehr langsam. QuantiPig dagegen, welches auf QuanPic aufbaut, hat zwar ebenfalls einen hohen Anspruch auf Speicher und CPU, kann die Performance aber deutlich steigern und erreicht mit knapp 26 MB Speicheranspruch eine geringere Belastung des Testgerätes. Wesentlich effizienter kommen in beiden Tests die APP QuanPic und QuantiPic davon. QuanPic hat trotz seines Speicherplatzbedarfs eine sehr geringe Speicher- und CPU-Belastung. Im Schnitt benötigt es gerade mal 7,44 MB Speicher und belastet den CPU dabei mit nur 30%. Die Performance ist dadurch sehr gut und ermöglicht ein ruckelfreies arbeiten mit der App. Auch QuantiPic hat mit durchschnittlich 10 MB Speicherbelastung einen sehr geringen Anspruch an das Testgerät. Die

**Analyseergebnisse Memory- und CPU-Belastung durch die APP's**

					Mittelwert					
<b>FotoQuant</b>	start	RGB	Midtread	Lloyd	59,12					
Memory (MB)	51,89	60,86	65,84	57,87	14,01					
free	16,00	16,00	16,00	8,03	45,11					
allocated	35,89	44,86	49,84	49,84	41,95					
CPU (%)	40,80	47,44	35,61	43,96	29,50					
User	22,80	24,79	27,32	43,10	12,45					
Kernel	18,00	22,65	8,29	0,86						
<b>QuanPic</b>	start / Ohne	MedianBlur			Mittelwert					
Memory (MB)	11,63	11,66			11,65					
free	3,88	4,54			4,21					
allocated	7,75	7,12			7,44					
CPU (%)	26,95	33,14			30,05					
User	24,78	28,49			26,64					
Kernel	2,17	4,65			3,41					
<b>QuantiPic</b>	start / Ohne	Helligkeit	Farbwerte	Menü	Mittelwert					
Memory (MB)	12,79	12,79	12,79	12,79	12,79					
free	4,96	2,97	2,66	0,49	2,77					
allocated	7,83	9,82	10,13	12,30	10,02					
CPU (%)	30,50	42,18	38,28	0,00	27,74					
User	26,64	40,14	32,72	0,00	24,88					
Kernel	3,86	2,04	5,56	0,00	2,87					
<b>QuantiPig</b>	start / Ohne	Skalar 1	Skalar 2	Skalar 4	Skalar 8	Pixel 1	Pixel 2	Pixel 4	Pixel 8	Mittelwert
Memory (MB)	11,74	19,61	19,61	19,62	19,62	46,64	27,32	21,54	19,62	25,73
free	4,55	7,84	7,84	7,85	7,85	16	10,92	8,61	7,56	9,80
allocated	7,19	11,77	11,77	11,77	11,77	30,64	16,4	12,93	12,06	15,93
CPU (%)	39,91	38,49	34,91	27,60	29,77	40,93	37,06	38,69	37,37	35,24
User	35,62	34,07	33,19	24,8	25,33	33,33	31,18	35,71	34,62	30,83
Kernel	4,29	4,42	1,72	2,8	4,44	7,6	5,88	2,98	2,75	4,41

Abbildung 2.4: Ergebnisse AndroidStudio Memory- und CPU-Monitor

CPU-Belastung ist dabei noch etwas kleiner gegenüber QuanPic. Mit knapp 27,8% lässt sich die APP sehr flüssig bedienen und das trotz des größeren Funktionsumfangs. Eine Übersicht über die Ergebnisse des Memory- und CPU-Monitors sehen Sie in Abbildung 2.4.

### 2.3 VERGLEICH DER VORHANDENEN DREI APPS

## Vergleich der drei vorhandenen APP's

	FotoQuant			QuanPic		QuantiPic		
Merkmale:	• drei Modi			• zwei Modi		• drei Modi		
Modus:	Original	Midtread	Lloyd	Original	MedianBlur	Original	Helligkeit	Farbwerte
Dateityp:	JPG	JPG	JPG	PNG	PNG	PNG	PNG	PNG
Abmessung:	2688 x 1520	2688 x 1520	2688 x 1520	1456 x 832	1456 x 832	1456 x 832	1456 x 832	1456 x 832
Dateigröße (KB):	1603	2401	2625	2035	1500	1822	1125	38
Kompression zum jeweiligen Original:	100%	-50%	-64%	100%	26%	100%	38%	98%
Vorteile:	<ul style="list-style-type: none"> <li>APP speichert Bilder je nach Verfahren in einem spezifischen Ordner</li> <li>setzt Zeitstempel bei Bildern</li> </ul>			<ul style="list-style-type: none"> <li>hat OpenCV bereits integriert</li> <li>zeigt Auflösung des Bildes live an</li> <li>zeigt Livebild flüssig an (14fps)</li> <li>zeigt fps an</li> </ul>		<ul style="list-style-type: none"> <li>Livebild bei Originalbild flüssig</li> <li>Livebild bei "Farbwerte" flüssig</li> <li>"Verarbeitete" Bilder sind kleiner als das Original(komprimiert)</li> </ul>		
Nachteile:	<ul style="list-style-type: none"> <li>sehr langsame Verarbeitung in Echtzeit</li> <li>langsame Verarbeitung bei Originalbild</li> <li>verarbeitete Bilder sind größer als Orig.</li> <li>bei Midtread wird nur die obere Hälfte des Bildes quantisiert</li> <li>OpenCV muss nachträglich auf dem Smartphone installiert werden</li> </ul>			<ul style="list-style-type: none"> <li>ruckelt stark bei "MedianBlur" (2fps)</li> <li>keine intuitive Bedienung</li> </ul>		<ul style="list-style-type: none"> <li>OpenCV muss nachträglich auf dem Smartphone installiert werden</li> <li>Livebild bei "Helligkeit" stockend</li> </ul>		
Usability:	<ul style="list-style-type: none"> <li>Quantisierung ist über Menü auswählbar</li> <li>besitzt einen Auslösebutton</li> <li>Hilfefunktion mit Beschreibung der einzelnen Menüpunkte</li> </ul>			<ul style="list-style-type: none"> <li>keine Erklärung der Funktionen</li> </ul>		<ul style="list-style-type: none"> <li>intuitive Bedienung</li> <li>Verfahren ist über ein Menü auswählbar</li> </ul>		

Tabelle 1: Vergleich der drei vorhandenen APP's

## Vergleich Standard-APP zu QuantiPig

	HTC	QuantiPig								
Merkmale:	• Standard-APP	• drei Modi								
Modus:	Original	Original	Pixel-1	Pixel-2	Pixel-4	Pixel-8	Skalar-1	Skalar-2	Skalar-4	Skalar-8
Dateityp:	JPG	PNG	PNG	PNG	PNG	PNG	PNG	PNG	PNG	PNG
Abmessung:	2688 x 1520	1456 x 832	1456 x 832	1456 x 832	1456 x 832	1456 x 832	1456 x 832	1456 x 832	1456 x 832	1456 x 832
Dateigröße (KB):	953	2054	2003	1375	727	371	60	166	546	2019
Kompression zum jeweiligen Original:	-	100%	2%	33%	65%	82%	97%	92%	73%	2%
Eigenschaften:	-	<ul style="list-style-type: none"> <li>• Quantisierung im Livebild</li> <li>• speichert Bilder je nach Verfahren in einem spezifischen Ordner <i>[FotoQuant]</i></li> <li>• setzt Zeitstempel bei Bildern <i>[FotoQuant]</i></li> <li>• zeigt Auflösung und fps des Bildes live <i>[QuanPic]</i></li> <li>• OpenCV bereits integriert <i>[QuanPic]</i></li> <li>• Modus ist über Menü auswählbar <i>[FotoQuant &amp; QuantiPic]</i></li> <li>• besitzt einen Auslösebutton <i>[FotoQuant &amp; QuantiPic]</i></li> </ul>								

Tabelle 2: Vergleich Standard-APP zu QuantiPig

## 2.4 BEISPIELBILDER



Abbildung 2.5: FotoQuant Originalbild



Abbildung 2.6: FotoQuant Lloyd-Modus

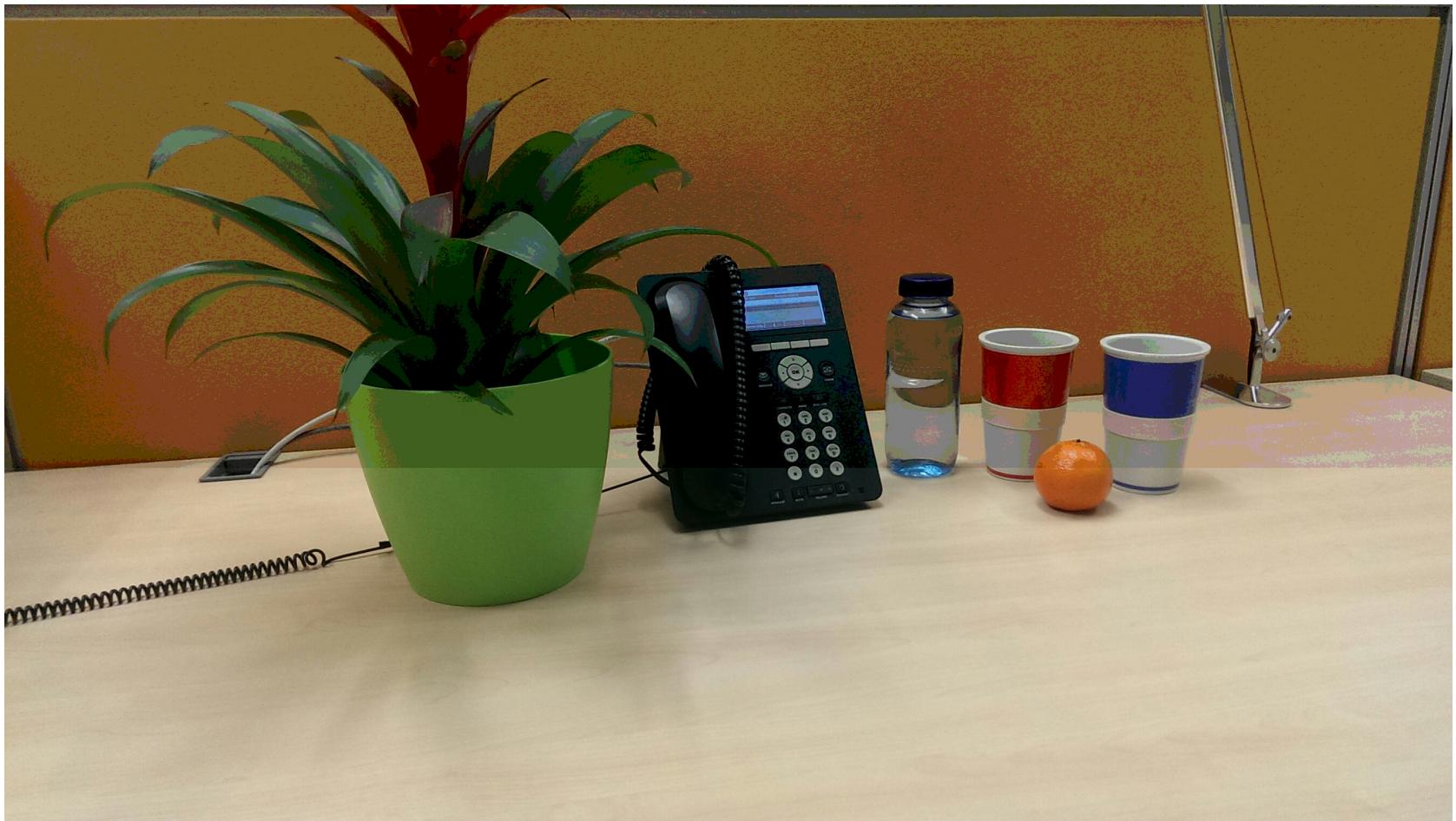


Abbildung 2.7: FotoQuant MidTread-Modus



Abbildung 2.8: Quanpic im Originalbild-Modus



Abbildung 2.9: Quanpic im MedianBlur-Modus



Abbildung 2.10: QuantiPic im Originalbild-Modus



Abbildung 2.11: QuantiPic im Helligkeits-Modus



Abbildung 2.12: QuantiPic im Farbwerts-Modus

## 2.5 PRODUKTFUNKTIONALITÄT

Auszug aus der Aufgabenbeschreibung:

*„Auf Basis der Analyse ist eine neue App zu programmieren, welch die positiven Eigenschaften der vorhandenen Apps vereint und als Verarbeitungsfunktion das Bild gleichmäßig mit einem einstellbaren Quantisierungsintervall quantisiert.“*

## 2.6 RANDBEDINGUNGEN

Der zeitliche Rahmen für die Entwicklung und Programmierung dieser APP endet mit der 4. Kalenderwoche 2016.

# 3 ANFORDERUNGEN

## 3.1 FACHKONZEPT

Die QuantiPig-APP wird in Java programmiert, um durch Verwendung bestehender Klassen die Erweiterbarkeit und Realisierbarkeit zu vereinfachen. Für das Design werden XML-Stylesheets verwendet.

### Verwendete Bibliotheken von Drittanbieteren

- OpenCV

## 3.2 INSTALLATIONSPROZEDUR

Die APP wird als .apk Datei ausgeliefert und kann somit manuell auf Android-Smartphones ab Version 4.4.2 installiert werden. Dabei muss das Installieren von Software mit unbekannter Herkunft erlaubt werden.

## 3.3 ANFORDERUNG AN DIE ENTWICKLUNG

### 3.3.1 ENTWICKLUNGS-UMGEBUNG

Für die Entwicklung wird Android Studio inkl. Gradle genutzt. Für die Dokumentation und Projektkoordination wird GitHub verwendet. Die Dokumentation wird mittels L<sup>A</sup>T<sub>E</sub>X erstellt.

### 3.3.2 ÄNDERUNGSMANAGEMENT

Zur Versionsverwaltung wurde Git eingesetzt. Als Hosting-Anbieter wurde dabei auf GitHub gesetzt, welcher einen kostenfreien Zugang für nicht kommerzielle Projekte bereitstellt. Ein Graphical User Interface (GUI) oder ein Konsolenprogramm für Windows und Linux übernehmen dabei die Steuerung der Versionsverwaltung. Konflikte in den einzelnen Versionen können nur über die Konsole behoben werden. Auf der Webseite von GitHub können Milestones erstellt werden und an die jeweiligen Mitarbeiter zugeteilt werden. In den Milestones

werden einzelne Aufgaben, sogenannte Issues angelegt und wiederum den Bearbeitern zugeordnet, somit ist der Bearbeitungsstand zu jeder Zeit des Projektes ersichtlich und es kann schnell auf sich ergebende Probleme reagiert werden.

## 4 ERGEBNIS

### 4.1 ÜBERNOMMENE FUNKTIONALITÄTEN

Basierend auf den Ergebnissen des App-Vergleichs, wurden die Funktionalitäten von folgenden Apps übernommen:

#### QuanPic

- Um eine zusätzliche Installation des OpenCV-Managers (neben der eigentlichen App) auf das Smartphone überflüssig zu machen, wurde der grundlegende Rahmen dieser App übernommen, da diese als einzige der drei vorgegebenen Apps jene Funktionalität unterstützt.
- Mit der Übernahme des Rahmens von “QuanPic“ wird im Live-Bild auch die Frame-Rate und die Displayauflösung angezeigt.

#### QuantiPic

- Aus dieser App wurde die grundlegende Menüführung übernommen. So werden auch in der App “QuantiPic“ die Auswahl der Modi und des Intervalls via *AlertDialog* abgefragt.

#### FotoQuant

- Das Speichern der Bilder in entsprechende Ordner, je nach ausgewähltem Verfahren, sowie die Übernahme des Zeitstempels des aufgenommenen Bildes wurde aus “FotoQuant“ übernommen.
- Der Modus “Midtread“ wurde grundlegend übernommen, jedoch modifiziert.

### 4.2 NICHT ÜBERNOMMENE FUNKTIONALITÄTEN

Da jede der vorgegebenen Apps jeweils 2 Modi (neben der Darstellung des Originalbildes) enthielt, wurde sich nach mehreren Tests gegen folgende Verfahren entschieden:

#### QuanPic

- MedianBlur
  - Ein Filterverfahren zur Kantenglättung aus der OpenCV-Bibliothek (s. [OpenCV-Dokumentation](#))
- NeuQuant

- Farbquantisierung auf Basis eines künstlichen neuronalen Netzes (s. [Kohonenetz\(Wikipedia\)](#))
- zu leistungsschwach

### QuantiPic

- Helligkeit
- Farbwerte
  - Farbquantisierung durch Festlegung von Schwellwerten
  - Vorerfertigte Methode aus der OpenCV-Bibliothek und damit kaum modifizierbar.(s. [OpenCV-Dokumentation](#))
  - Die Implementierung, so wie sie in QuantiPic ist, zeigt ein Negativ mit 8 Farben.

### FotoQuant

- Lloyd
  - Vektorquantisierung auf Basis des Lloyd-Algorithmus (s. [K-Means-Algorithmus\(Wikipedia\)](#))
  - zu leistungsschwach

## 4.3 EIGENE ENTWICKLUNGEN

Da laut Aufgabenstellung eine Quantisierung mit **einstellbarem Intervall** gefordert war und dies in keiner der drei vorgegebenen Apps realisiert wurde, musste hier ein eigenes Verfahren implementiert werden. (s. *setCluster*)

## 4.4 UMFANG DER APP



Abbildung 4.1: QuantiPig Logo

Die App wurde, angelehnt an die Namen der vorhandenen Apps, "QuantiPig" genannt. Als Logo wurde ein Schweinekopf gewählt.(siehe Abbildung 4.1)

Als Algorithmen wurden folgende Verfahren implementiert:

- keine Quantisierung
- Pixel
- Skalar

Die Modi “Pixel“ und “Skalar“ können jeweils mit folgenden auswählbaren Intervallen verändert werden:

- 1
- 2
- 4
- 8

#### 4.5 ERSCHEINUNGSBILD

Die APP startet generell im Landscape-Modus. Wie in Abbildung 4.2 zu sehen, gibt es jeweils einen Button für die Wahl des Quantisierungsverfahrens und den Auslöser. Eine kleine Anzeige informiert über die aktuelle Bildrate und die Auflösung



Abbildung 4.2: QuantiPig Startbildschirm

Nach Drücken auf den Auslöser wird ein Foto mit aktuellem Verfahren gemacht und in den entsprechenden Ordner auf dem Smartphone abgelegt. Als Name wird das jeweilige Verfahren, gefolgt von einem Zeitstempel verwendet.

Nach betätigen des Buttons „Modus“, öffnet sich, wie in Abbildung 4.3 zu sehen, ein Auswahlmenü für die drei verschiedenen Verfahren.

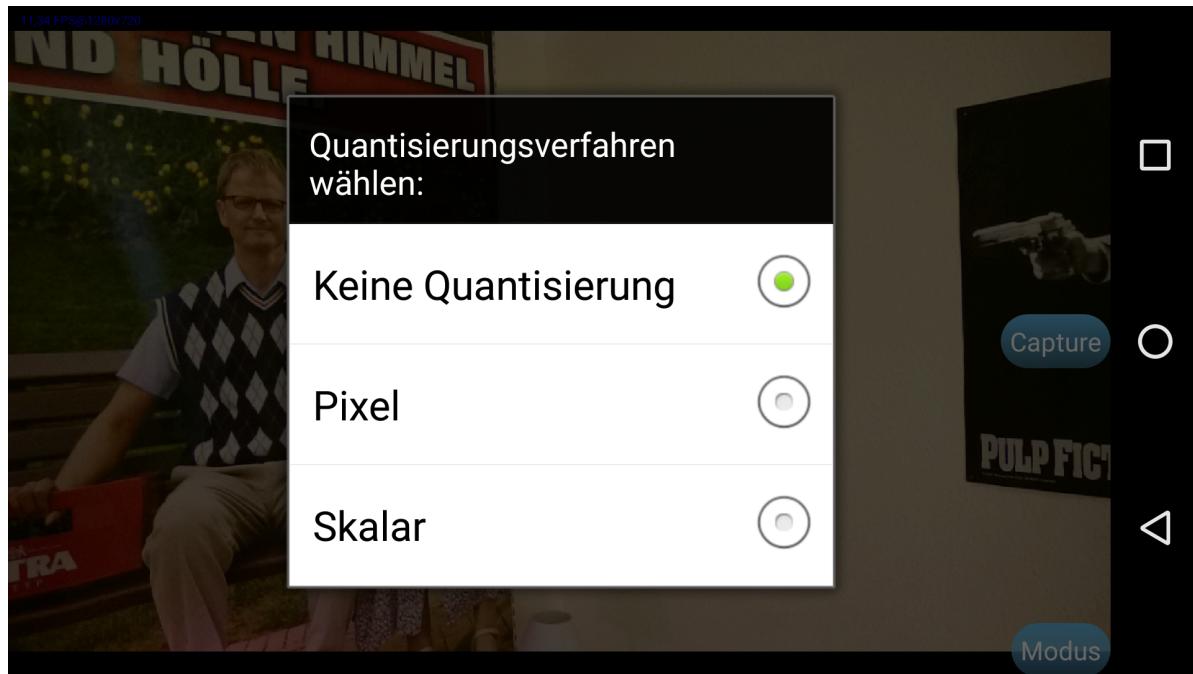


Abbildung 4.3: QuantiPig-Menü Verarbeitungsverfahren

Im Modus „Pixel“ und „Skalar“ (siehe Abbildung 4.4) gibt es einen weiteren Button mit dem man das Intervall auswählen kann.

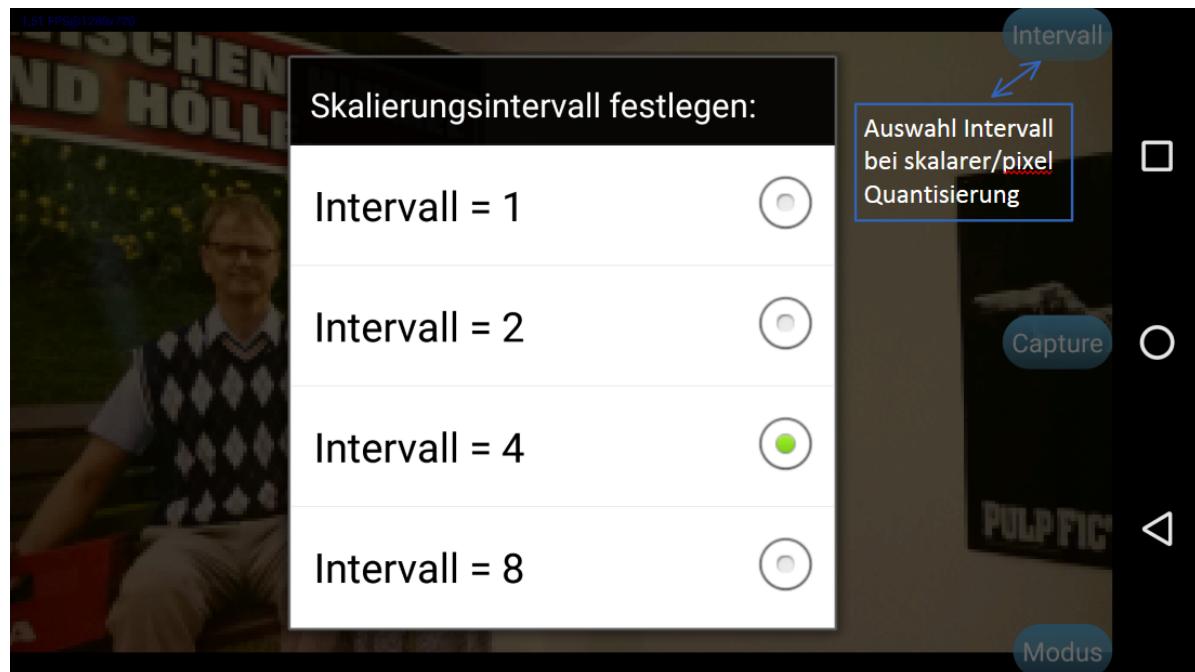


Abbildung 4.4: QuantiPig-Menü Quantisierungsintervall

## 4.6 QUELLCODE

### 4.6.1 ALLGEMEIN

Wie bereits erwähnt wurde der grundlegende Aufbau aus der App “QuanPic“ übernommen. Neben der Hauptklasse wurde für die beiden Bearbeitungsmodi jeweils eine eigene Klasse erstellt:

- MainActivity.java
- Pixel.java
- Skalar.java

Im Folgenden wird nicht auf jedes Detail im Quellcode eingegangen, jedoch die wichtigsten Funktionen erläutert.

### 4.6.2 MAINACTIVITY.JAVA

Die MainActivity ist die Haupt- und damit Startaktivität der App. Hauptaufgabe der Aktivität ist es, Objekte (Strings, Buttons, Menüs, Ladebalken etc.) beim Start der App aufzurufen und ggf. mit Aktions-Listener zu versehen, sofern die Objekte hier auf Nutzereingaben reagieren sollen.

Dies geschieht innerhalb der Methode “*onCreate()*;“, die das Layout einer View zuordnet und anschließend bereits erwähnte Objekte instanziert.

Folgender Ausschnitt zeigt am Beispiel von mCameraView, wie dieses einem Layoutobjekt (*surface\_view*) zugeordnet wird, die Sichtbarkeit manipuliert wird und mit dem CameraListener bestückt wird:

```
setContentView(quantipig.R.layout.activity_main);
mCameraView = (CameraBridgeViewBase) findViewById(R.id.
    surface_view);
```

Darüber hinaus sind hier einige wichtige Methoden aufgeführt, die essentiell für die Funktionsweise von OpenCV und somit für die App als funktionierende Smartphone-Kamera sind:

- onResume()
- BaseLoaderCallback()
- onPause()
- onDestroy()
- onStart()

Weitere Details können auch der [OpenCV-Dokumentation](#) entnommen werden.

## imageProcessing()

Diese Methode ist das Kernstück für die Bildbearbeitung der App. Hier sind auch die Standard-Methoden von OpenCV implementiert: s. hierzu OpenCV-Dokumentation:

- [CvCameraViewListener2](#)
- [CvCameraViewFrame](#)

Die Methode “*onCameraFrame()*“ erhält Einzelbilder von der Kamera und übergibt dieses an ein Objekt der Open-CV Klasse “*Mat*“ als RGBA-Matrix. Diese Matrix kann anschließend bearbeitet werden und wird anschließend weiter an das Display des Endgeräts gegeben. Der grundlegende Aufbau wurde aus der Vorlage “QuantPic“ übernommen. Es wurden jedoch diverse Änderungen am Quelltext vorgenommen. Die wichtigsten Änderungen:

- Speichern der Höhe, der Breite und der Anzahl der Kanäle des Live-Bildes, damit diese Werte an den jeweiligen Modus übergeben werden können.

```
Mat mMat;
int mHeight = inputFrame.rgba().height();
int mWidth = inputFrame.rgba().width();
int channels = inputFrame.rgba().channels();
```

- Änderung der Abfrage des ausgewählten Modus von einer if- zu einer **switch case** Anweisung. Dabei wird der Wert der Variable **modeSelector** übergeben, welche wiederum den ausgewählten Modus repräsentiert und somit die Ausgabe auf dem Display entsprechend ändert.

```
switch (modeSelector){
    case 0: (...) /* keine Quantisierung */
    case 1: (...) /* Pixel */
    case 2: (...) /* Skalar */
}
```

- Abfrage des ausgewählten Intervalls (nur bei “Pixel“ und SSkalar“)

```
cluster = setCluster();
```

- Übergabe des Live-Bildes und den ermittelten Variablen an die jeweilige Klasse zur Bearbeitung (nur bei “Pixel“ und SSkalar“)

```
mMat = Pixel.pixel(inputFrame.rgba(), mHeight, mWidth,
    channels, cluster);
```

- Übergabe des bearbeiteten Bildes zum Speichern

```
getsavingImage(mMat);
```

- Übergabe des bearbeiteten Bildes an das Display

```
return mMat;
```

### getsavingImage()

Diese Methode speichert das aus der Methode `imageProcessing()` übergebene Bild und stellt es der Methode `saveImage` zur Verfügung, sofern der Button zum Speichern betätigt wurde.

```
public void getsavingImage(Mat mat) {
    previewImage = mat;
}
```

### setModus()

Diese Methode baut die Abfrage des Modus via AlertDialog und OnClickListener auf und übergibt entsprechend der Auswahl einen Wert an die Variable `modeSelector`, welche wiederum von der Methode `imageProcessing()` ausgelesen wird.

Die Abfrage erfolgt wiederum über eine switch case Anweisung. Ausschlaggebend ist der Wert der Variable `quantizationMode`.

Je nach ausgewähltem Modus wird noch der Intervall-Button aus- (*Keine Quantisierung*) oder eingeblendet (*Pixel, Skalar*).

```
switch (quantizationMode) {
    case QUANT_MODE_0: { /* Keine Quantisierung */
        hideButton();           /* Verstecke Intervall-Button */
        modeSelector = 0;
        break;
    }
    case QUANT_MODE_1: { /* Pixel */
        showButton();          /* Zeige Intervall-Button */
        modeSelector = 1;
        break;
    }
    ...
}
```

### setCluster()

Diese Methode übergibt anhand des ausgewählten Eintrags in der Methode `createClusterMenu()` den entsprechenden Intervall-Wert in die Variable `cluster`.

Diese Intervalle werden dann von den Klassen “Pixel” und “Skalar” verwendet um unterschiedliche Ergebnisse zu erzielen.

Folgende Intervall-Werte gibt es:

- 1
- 2
- 4
- 8

Als Abfragewert dient die Variable `selectedCluster` aus der Methode `createClusterMenu()`.

```

switch (selectedCluster) {
    case Cluster_0: {
        cluster = 1;
        return cluster;
    }
    ...
}

```

### createClusterMenu()

Hier wird die Abfrage des Intervalls mittels AlertDialog realisiert. Der Ausgewählte Wert wird schließlich von der Methode `setCluster` verwendet.

### saveImage()

Sobald der Button “Capture“ betätigt wurde, wird das (bearbeitete) Bild in diese Methode übergeben und zum Speichern als .png-Datei vorbereitet.

Dabei wird zunächst der aktuelle Zeitpunkt ermittelt und gespeichert.

```

SimpleDateFormat sdf = new SimpleDateFormat("yy_MM_dd-HH_mm_ss");
String currentDateAndTime = sdf.format(new Date());

```

Anschließend wird das Bild von RGBA in einen BGRA-Farbraum konvertiert, da sonst das gespeicherte Bild vom angezeigten Bild auf dem Display abweicht (Blau wird als Rot dargestellt und umgekehrt).

```
Imgproc.cvtColor(mat, mat, Imgproc.COLOR_RGBA2BGRA, 4);
```

Entsprechend des ausgewählten Modus und Intervalls wird nun der Pfad- und Dateiname erstellt.

Alle Bilder werden unter DCIM/QuantiPig/ gespeichert.

Am Beispiel für Modus 1 (Pixel):

```

rootPath = new File(Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_DCIM).getPath() + "/QuantiPig/Pixel");
fileName = QUANT_MODE_1_STRING + "-" + cluster + "_" +
    currentDateAndTime + ".png";

```

Als Resultat wird das Bild in den Ordner *DCIM/QuantiPig/Pixel* gespeichert. Der Dateiname lautet dann beispielsweise: *Pixel-8\_15\_12\_24-18\_59\_30.png*

#### 4.6.3 PIXEL.JAVA

Diese Klasse dient als Container für die Methode *pixel()*.

Die Methode geht dabei wie folgt vor:

- Es werden Quadrate der Kantenlänge = Intervall auf das Originalbild gelegt.
- Der durchschnittliche Farbwert aller Bildpunkte innerhalb des Quadrats wird gebildet
- Die Bildpunkte des Quadrates werden nun mit dem Durchschnittsfarbwert überschrieben.

Bei (Intervall > 1) entsteht der Eindruck, dass die Auflösung des Displays gesenkt wird und somit das Bild “verpixelt” wird. Bei steigendem Intervall wird der Effekt deutlicher.

#### pixel

Der äußere Rahmen der Methode. Es wird das Bild der Kamera, seine Maße, die Anzahl der Kanäle und das ausgewählte Intervall übergeben.

```
public static Mat pixel(Mat mat, int mHeight, int mWidth, int
    channels, int intervall){
    ...
    return mat;
}
```

Zur einfacheren Handhabe wird das übergebene Bild in ein Byte-Array geschrieben.

```
byte[] buff = new byte[mHeight * mWidth * channels];
mat.get(0, 0, buff);
```

Zunächst werden einige Variablen mit 0 instanziert.

```
int x = 0;
int k = 0;
int countY = 0;
int[] frame = new int[mWidth / cluster * mHeight / cluster *
    channels];
```

Kurze Eräuterung:

- x: Index des abzutastenden Farbwertes auf buff
- k: Index des RGBA-Bildpunktes des Hilfsarrays frame
- countY: Zähler, wie viele Zeilen von buff schon abgetastet wurden
- frame: Das Hilsarray zur Berechnung des Durchschnitts.

Nun beginnt das Abtasten und das Überschreiben der Werte aus buff in frame mit Hilfe von vier ineinander verschachtelten Schleifen.

Die äußere Schleife geht vertikal, also zeilenweise das Bild runter:

```
for (int n = 0; n < mHeight; n++) {
```

Es ist notwendig die Anzahl abgetasteten Zeilen zu kennen. Daher wird jeder Schleifendurchlauf der äußeren Schleife mitgezählt.

```
    countY++;
```

Nun beginnt der Schleifendurchlauf in der Horizontalen. Als erstes wird jedes Cluster einzeln durchlaufen.

```
for (int m = 0; m < mWidth / intervall; m++) {
```

Danach wird jeder Bildpunkt im Cluster durchlaufen.

```
for (int j = 0; j < intervall; j++) {
```

Und nun werden die RGBA-Kanäle des Bildpunktes durchlaufen.

```
for (int i = 0; i < channels; i++) {
```

Zusammenfassung:

- x = Index des Quell-Arrays
- k = Index des Ziel-Arrays
- n = Zeile des Quell-Arrays
- m = Index des Clusters in der Zeile
- j = Index des Bildpunktes innerhalb des Clusters
- i = Index des RGBA-Kanals im Bildpunkt

Damit können nun alle Koordinaten durchlaufen werden. Die Sache wird nur noch dadurch komplizierter, da JAVA mit signed-Werten arbeitet und sich somit der Wert inhaltlich ändert. (Wertebereich ändert sich von (0 bis 255) auf (-128 bis 127))

Daraus folgend werden Additionen mit Werten > 127 als Subtraktion ausgeführt. Dies wird vor allem dann an Kanten sichtbar, wo stark unterschiedliche Werte dicht beieinander liegen. Dieser Effekt wird ggf durch ein großes Intervall verstärkt. (s. linkes Bild von 4.5)

Um das Problem zu umgehen und das Resultat wie im rechten Teil des Bildes zu erzielen, werden die negativen Werte umgerechnet.

```
if (buff[x + i + j * channels] >= 0)
    frame[k + i] += ((int) buff[x + i + j * channels]);
else
    frame[k + i] += ((int) buff[x + i + j * channels] & 0xff);
```

Mit dem Schließen der inneren beiden Schleifen wurde das erste Cluster ( $m=0$ ) in der ersten Zeile vollständig abgetastet der erste Bildpunkt des Hilfsarray (zunächst) fertig befüllt. Daher wird der Index des Zielarrays um einen Bildpunkt, also um 4 (aufgrund der vier Kanäle) erhöht. Gleichzeitig springt der Index des Quell-Arrays auf den ersten Bildpunkt der in das neue Cluster geschrieben werden soll. Dies wird so lange gemacht, bis das Zeilenende erreicht wurde.

```
k += channels;
x += channels * intervall;
}
```

Dies wird nun Zeile für Zeile so oft wiederholt. Da der Algorithmus auch Bildpunkte eines Cluster abtasten muss, die untereinander liegen, muss nach jedem Zeilendurchlauf geprüft werden, ob die Clusterbreite ( $/$ -höhe) bereits erreicht wurde. Ist dies nicht geschehen, werden alle untereinander stehenden Werte in den selben Index des Ziel-Arrays geschrieben, womit dieser immer wieder am Zeilenanfang zurückgesetzt werden muss.

```
if (countY < intervall)
    k -= mWidth * channels / intervall;
```

Wurden nun auch in der Vertikalen alle Bildpunkte abgetastet, dürfen sich alle Indexe erhöhen. Nur das Zählen der Schleifendurchläufe, bis wieder die Clusterhöhe erreicht wurde, muss von vorne beginnen.

```
else
    countY = 0;
}
```

Nach dem nun alle Werte in das cluster-weise übertragen und im Hilfsarray aufsummiert wurden, muss nur noch der Durchschnitt berechnet werden (Summe der Elemente : Anzahl der Elemente). Bei der Methode muss man sich jedoch im Klaren sein, dass hier aufgrund des verwendeten Datentyps **integer** eine Ganzzahldivision stattfindet, was am Ende zu Rundungsfehlern führt.

```
for (int i = 0; i < frame.length; i++) {
    frame[i] = frame[i] / (intervall * intervall);
}
```

Nun werden die Durchschnittswerte wieder zurück in das Quell-Array geschrieben. Dabei werden die gleichen vier Schleifen wie zuvor durchlaufen.

```
k = 0;
countY = 0;
x = 0;

for (int n = 0; n < mHeight; n++) {
    countY++;

    for (int m = 0; m < mWidth / intervall; m++) {
```

```
    for (int j = 0; j < intervall; j++) {
        for (int i = 0; i < channels; i++) {
            buff[x + i + j * channels] = (byte) frame[k + i];
        }
    }
    k += 4;
    x += 4 * intervall;
}
if (countY < intervall)
    k -= mWidth * channels / intervall;
else
    countY = 0;
}
```

Zum Schluss wird das Byte-Array wieder zurück in das Bild geschrieben.

```
mat.put(0, 0, buff);
```



Abbildung 4.5: Gleiches Motiv mit einem 8 x 8 Intervall aufgenommen

#### 4.6.4 SKALAR.JAVA

Diese Klasse dient als Container für die Methode “skalar“. Hier werden die binären Farbwerte mittels logischer Konjunktion (AND) mit einem binären Vektor  $v$  verknüpft. Der Anzahl der daraufhin dargestellten Farben  $n$  hängt vom gewählten Intervall  $i$  ab.  
 $n = 2^{i^k}$ , mit  $k = \text{Anzahl der Kanäle}$

Daraus folgt, bei  $k = 3$ (RGB):

- Intervall = 1:
  - $v = 1000\ 0000$
  - $n = 2^{1^3} = 8$
- Intervall = 2:
  - $v = 1100\ 0000$
  - $n = 2^{2^3} = 64$
- Intervall = 4:
  - $v = 1111\ 0000$
  - $n = 2^{4^3} = 4096$
- Intervall = 8:
  - $v = 1111\ 1111$
  - $n = 2^{8^3} = 16777216$

#### skalar

Der Rahmen dieser Methode ist analog zu `pixel()` aufgebaut. Es werden das Originalbild (als RGBA-Matrix), die Maße, die Anzahl der Kanäle, sowie das eingestellte Intervall übergeben, in ein Byte-Array geschrieben und am Ende das modifizierte Byte-Array in das Bild geschrieben, welches dann wieder zurückgegeben wird.

```
public static Mat skalar(Mat mat, int mHeight, int mWidth, int
    channels, int intervall) {
    byte[] buff = new byte[mHeight * mWidth * channels];
    mat.get(0, 0, buff);

    (...)

    mat.put(0, 0, buff);
    return mat;
}
```

Der innere Aufbau dieser Methode ähnelt dem Modus “Midtread“ der App “Foto-Quant“. Es wurden jedoch einige Änderungen vorgenommen.

Zunächst wird ein Bitshiftvektor initialisiert. Da Java Bitshift-Operationen nur mit dem Datentyp `Integer` (32 Bit) durchführen kann, das Byte-Array jedoch 8 Bit pro Index enthält, werden zunächst alle Bit  $B_i$  mit einem Index  $i > 7$  mit `1` initialisiert.

```
int bitshift = 0xFFFFFFFF00;
```

Danach werden die Bits in Abhängigkeit vom Intervall nach rechts verschoben (shift-right Operator)

```
bitshift = bitshift >> intervall;
```

Nun kommt der Aufbau, der aus "FotoQuant" übernommen wurde. Das Array wird abgetastet und jeder einzelne Farbwert wird modifiziert. Allerdings ist diese Modifizierung selbst wiederum anders, als in der Vorlage.

```
int t;
for (int i = 0; i < mWidth * mHeight; i++) {
    t = i * channels;                                /* Bildpunkte */
    buff[t] = (byte) (buff[t] & bitshift); /* Rot */
    buff[t + 1] = (byte) (buff[t + 1] & bitshift); /* Gelb */
    buff[t + 2] = (byte) (buff[t + 2] & bitshift); /* Blau */
}
```

#### 4.7 BEISPIELBILDER

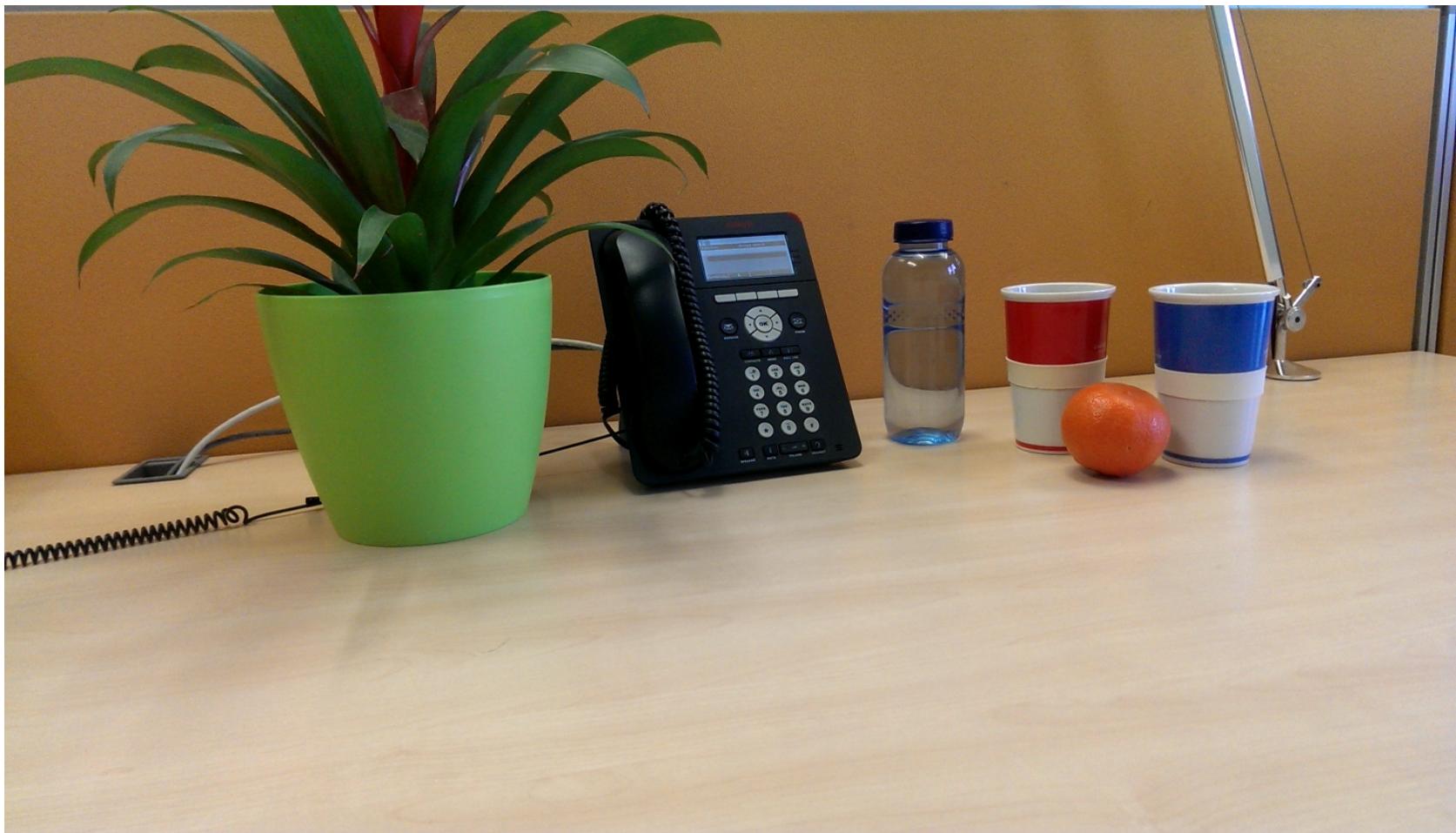


Abbildung 4.6: QuantiPig Originalbild

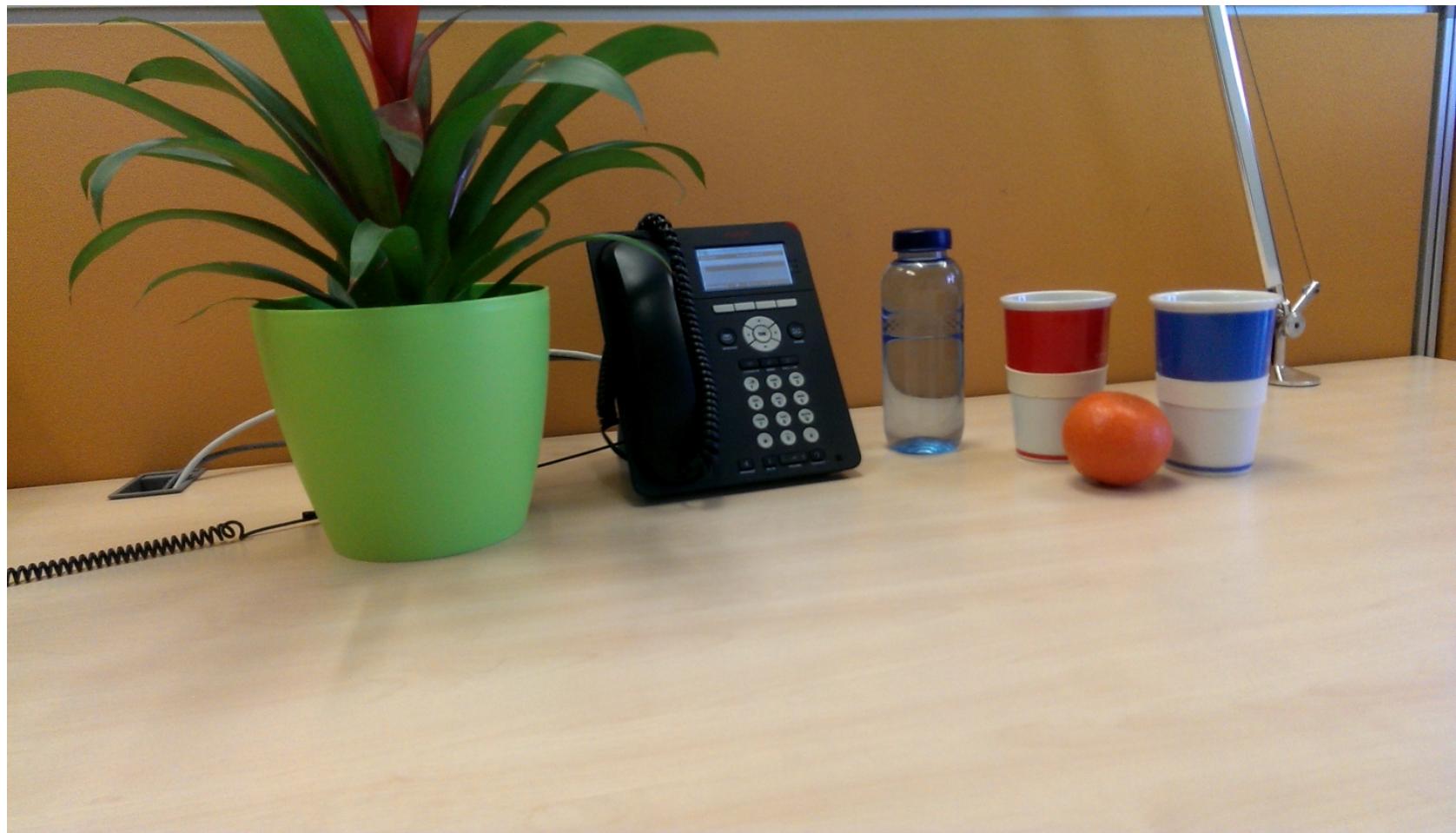


Abbildung 4.7: QuantiPig Pixel Modus Intervall 1



Abbildung 4.8: QuantiPig Pixel Modus Intervall 2

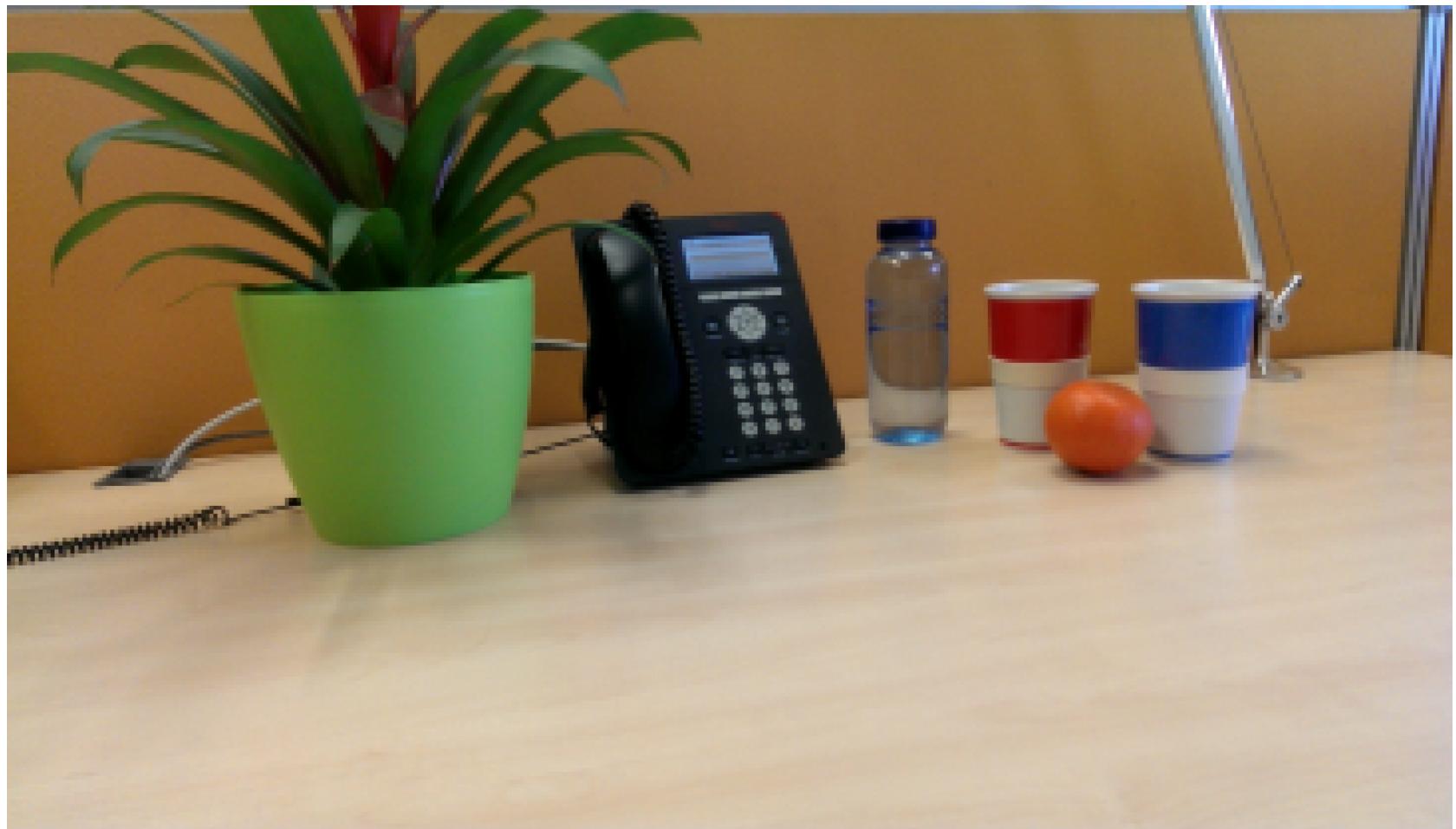


Abbildung 4.9: QuantiPig Pixel Modus Intervall 4

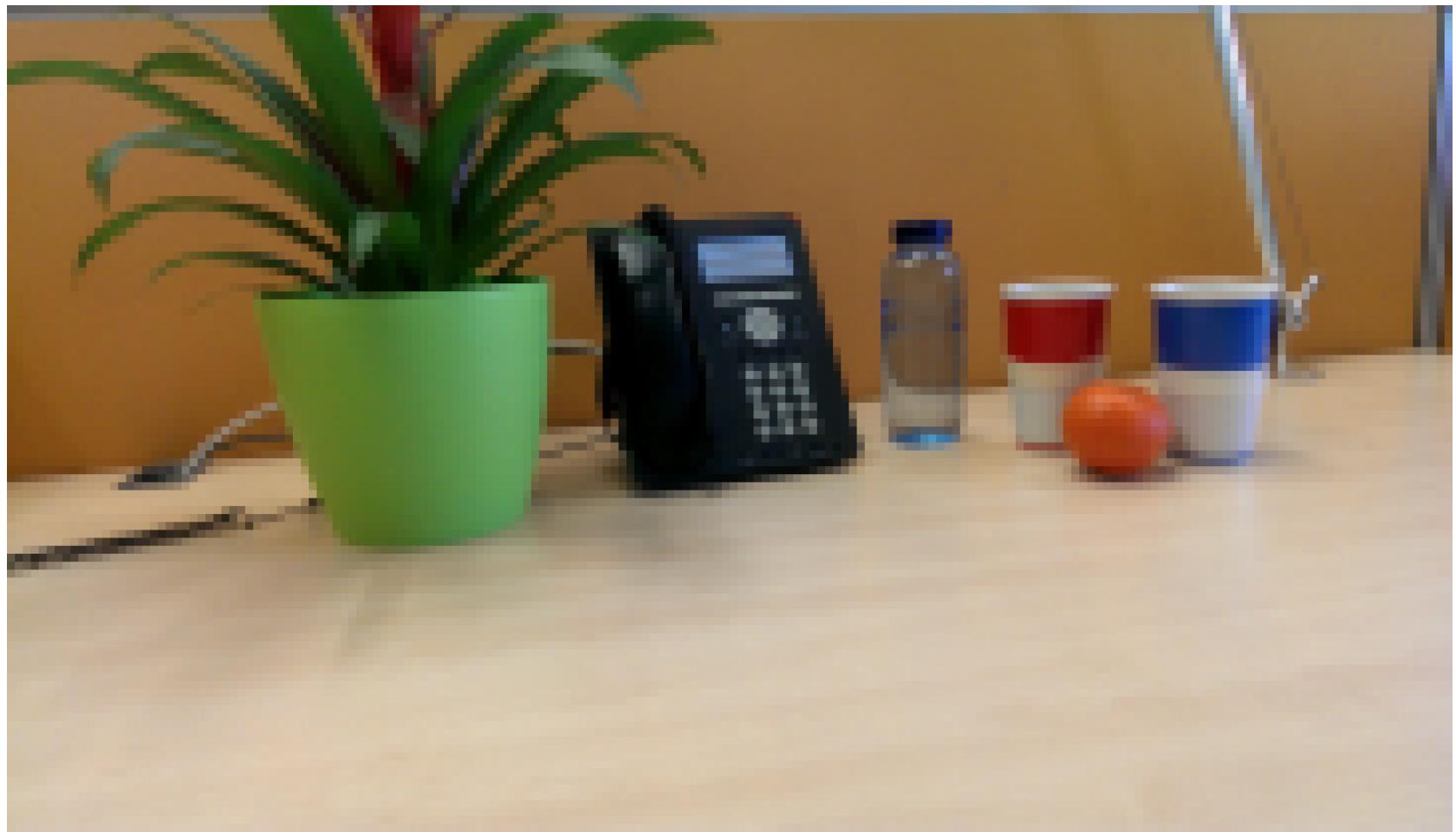


Abbildung 4.10: QuantiPig Pixel Modus Intervall 8



Abbildung 4.11: QuantiPig Skalar Modus Intervall 1



Abbildung 4.12: QuantiPig Skalar Modus Intervall 2

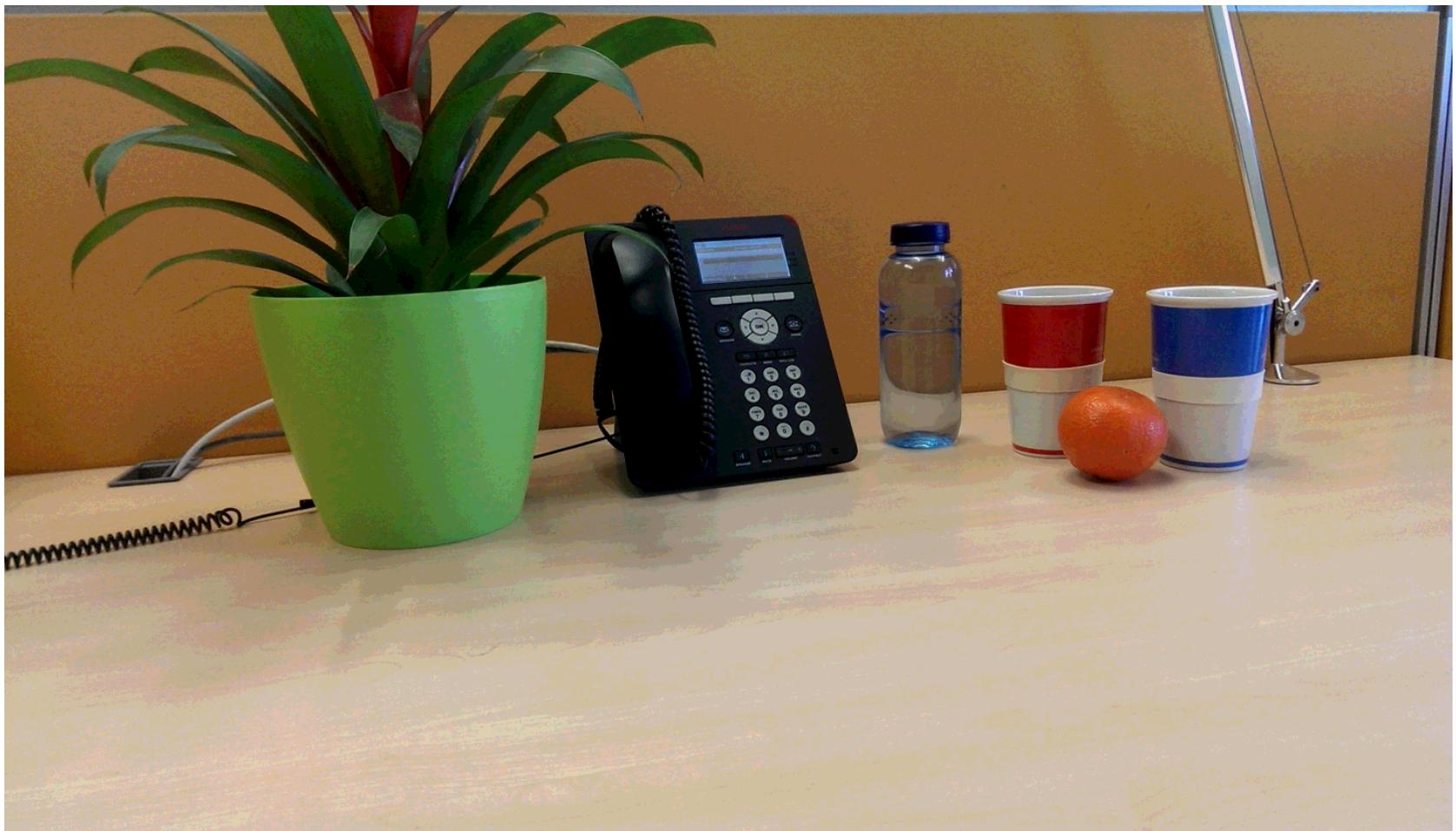


Abbildung 4.13: QuantiPig Skalar Modus Intervall 4



Abbildung 4.14: QuantiPig Skalar Modus Intervall 8

## 5 ANHANG

## 5.1 PVL AUFGABENBESCHREIBUNG

Hochschule für Telekommunikation Leipzig  
Institut für Kommunikationstechnik  
Gustav-Freytag-Straße 43-45  
04277 Leipzig

## Themen zur PVL IKT-KMI-13

Hinweise:

- ein Thema auswählen
- Mitstreiter benennen (inklusive Emailadresse) und deren (Teil-)Aufgabe(n)
- Name des Teams wählen (Ort der Dienststelle, wenn möglich)
  - Emails ohne Angabe „PVL, StudiengangXX, Team NameXX“ werden von mir ignoriert!!
- Max. 3 Teams pro Thema (first come first serve)
- Bearbeitungszeitraum bis 4. KW, Verzögerung führt zu Punktabzug
- Formatierung der Quelltexte, siehe: <http://www1.hft-leipzig.de/ice/Files/c-quell.txt>
- strukturelle und inhaltliche Gestaltung der Dokumentation gemäß
  - <http://www1.hft-leipzig.de/ice/Files/ThesisTemplate.zip>
- Für Programmieraufgaben darf **keine neuere** Version als Visual C++ 2008 Express verwendet werden.
- Genaue Hinweise (+Präzisierung der Aufgabenstellung, Unterlagen, Daten) gibt es nach Wahl eines Themas.

### "Modifikation eines Prädiktors (MED) in Anwendung auf Farbbilder"

Es ist eine Software zur Bilddatenkompression vorhanden (TSIPcoder). Über eine grafische Nutzerschnittstelle (GUI) kann Einfluss auf die Verarbeitungskette genommen werden. Soll ein Farbbild komprimiert werden, so werden die RGB-Komponenten typischer Weise in einen anderen Farbraum (YUV) konvertiert. Dadurch werden die drei Komponenten dekorreliert. Trotzdem enthalten die Komponenten Y, U und V noch ähnliche Strukturen, insbesondere hinsichtlich der Richtung der Kanten.

Der Median-Edge-Detection Prädiktor (MED) nutzt Kanteninformation aus, um zwischen drei verschiedene Prädiktoren umzuschalten. In der vorliegenden Version wird das Umschalten für alle drei Komponenten separat durchgeführt. Leider wird die Entscheidung manchmal durch Rauschen im Bild ungünstig beeinflusst.

Aufgabe ist es, eine zweite Version des MED-Prädiktors zu implementieren, bei der die Prädiktorauswahl für U und V auf Basis von Informationen aus Y erfolgt. Wenn die Y-Komponente bereits verarbeitet ist (dem Decoder steht die Information über Y zur Verfügung), dann kann nachträglich geprüft werden, ob für einen konkreten Bildpunkt der ausgewählte Prädiktor der beste war oder einer der beiden anderen evtl. einen kleineren Schätzfehler ergeben hätte. Die beste Variante wird dann auch für U und V eingesetzt.

Für einen vorgegebenen Satz an verschiedenen Bildern ist die neue Variante zu testen und die Ergebnisse mit dem normalen MED-Prädiktor zu vergleichen.

Der Quellcode ist klar zu strukturieren, mit ausreichenden Kommentaren zu versehen und gemäß

den Richtlinien zu formatieren. Variablenamen sollten selbsterklärend sein. Alle Untersuchungen sind schriftlich zu dokumentieren. Neben der schriftlichen Arbeit sind alle Quellen (Programmcode, Texte, Testbilder) und Tools abzugeben, damit eine Reproduktion der Ergebnisse möglich ist.

Teilaufgaben:

- Koordination
- Programmierung
- Dokumentation (Grundlagen, Änderungen am Quellcode, Kompressionsergebnisse)

Max. 4 Personen,

Max 2 Zusatzpunkte für Klausur

#### **"Programmieren einer (Smartphone-)App zur einfachen Kompression von Bildern"**

Es existieren drei Android-App, welche ein Bild von der Smartphone-Kamera im Roh-Format aufnehmen, verarbeiten und speichern. Diese drei Programme sind zu vergleichen und Vor- und Nachteile hinsichtlich verschiedener Eigenschaften (z.B. Bedienkomfort, Speicherbedarf, Bildfolgefrequenz, ...) zu ermitteln. Auf Basis der Analyse ist eine neue App zu programmieren, welche die positiven Eigenschaften der vorhandenen Apps vereint und als Verarbeitungsfunktion das Bild gleichmäßig mit einem einstellbaren Quantisierungsintervall quantisiert. Um die Bildfolgefrequenz zu erhöhen ist ggf. die Ausgabe-Bildgröße zu verringern.

Alle Untersuchungen sind schriftlich zu dokumentieren. Neben der schriftlichen Arbeit sind alle Quellen (Programmcode, Texte, Testbilder) und Tools abzugeben, damit eine Reproduktion der Ergebnisse möglich ist.

Teilaufgaben:

- Koordination
- Recherche
- Programmierung
- Dokumentation (Grundlagen, Methode, Änderungen am Quellcode, Kompressionsergebnisse)

Max. 4 Personen

Max. 2 Zusatzpunkte für Klausur

#### **"Analyse Template-Matching-Prädiktion im CoBaLP2-Coder"**

CoBALP2 ist ein Verfahren für die kontextbasierte lineare Prädiktion von Bilddaten auf Basis von Differenzwerten. Die Gewichte für die Berechnung der Schätzwerte werden für automatisch und signalangepasst festgelegte Kontexte sukzessive optimiert. Für manche Kontexte ist die Streuung der Schätzfehler jedoch relativ hoch. Hier kann evtl. eine Template-Matching-Prädiktion erfolgreich sein. Als Parameter sind Template-Größe, Größe des Suchraumes und vor allem die Verknüpfungsmethode von verschiedenen Matches einzustellen. Je nach Bildinhalt

können verschiedene Einstellungen günstig sein

Aufgabe ist es, systematisch zu testen, welche Einstellungen für welches Bild optimal sind. Daraus ist abzuleiten, wie die optimalen Einstellungen automatisch gewählt werden können.

Der Quellcode ist klar zu strukturieren, mit ausreichenden Kommentaren zu versehen (inkl. Tags zum Markieren der Änderungen am originalen Quellcode) und gemäß den Richtlinien zu formatieren. Variablennamen sollten selbsterklärend sein. Alle Untersuchungen sind schriftlich zu dokumentieren. Neben der schriftlichen Arbeit sind alle Quellen (Programmcode, Texte, Testbilder) und Tools abzugeben, damit eine Reproduktion der Ergebnisse möglich ist.

Teilaufgaben:

- Koordination
- Recherche
- Programmierung
- Dokumentation (Grundlagen, Methode, Änderungen am Quellcode, Kompressionsergebnisse)

Max. 4 Personen,

Max. 2 Zusatzpunkte für Klausur

## **"Restaurierung von synthetischen Bilddaten nach DCT-basierter JPEG-Kompression durch near-lossless Codierung"**

Synthetische Bilddaten werden aus Unkenntnis häufig mit JPEG (DCT-basierter Modus) komprimiert. Dadurch entstehen störende rauschartige Strukturen im Bild, insbesondere an Helligkeits- oder Farbkanten.

Eine vorhandene Software zur verlustlosen Bilddatenkompression (TSIPcoder) soll so modifiziert werden, dass auf Basis einer gewissen Toleranz das Rauschen im Bild reduziert wird. Dadurch wird die Kompression verlustbehaftet. Ein entsprechender Parameter ist in die grafische Nutzerschnittstelle (GUI) aufzunehmen.

Der Quellcode ist klar zu strukturieren, mit ausreichenden Kommentaren zu versehen (inkl. Tags zum Markieren der Änderungen am originalen Quellcode) und gemäß den Richtlinien zu formatieren. Variablennamen sollten selbsterklärend sein. Alle Untersuchungen sind schriftlich zu dokumentieren. Neben der schriftlichen Arbeit sind alle Quellen (Programmcode, Texte, Testbilder) und Tools abzugeben, damit eine Reproduktion der Ergebnisse möglich ist.

Teilaufgaben:

- Koordination
- Programmierung
- Dokumentation (Grundlagen, Methode, Änderungen am Quellcode, Kompressionsergebnisse)

Max. 4 Personen,  
Max. 2 Zusatzpunkte für Klausur

## **"Optimierung von Parametern zur Kompression von Farbbildern"**

Eine vorhandene Software (TSIP) zur Kompression von Bilddaten analysiert das geladene Bild und wählt günstige Parameter zur Kompression aus. Die Auswahl ist jedoch nicht in jedem Fall optimal.

Für ein gegebenes Set von Bilddaten sind manuell die Einstellungen zu variieren um eine bessere Kompression zu erzielen. Auch das Decodieren der komprimierten Bilder ist zu testen. Das TSIP-Programm kann über ein Batch-File aufgerufen, sodass ein systematischer Test mit einer Vielzahl von verschiedenen Einstellungen erfolgen kann.

Die Zusatzpunkte ergeben sich wie folgt:

- besseres Kompressionsergebnis als mit automatisch gewählten (oder bereits bekannten) Parametern: 0.05 Punkte (pro Team)
- -" und besseres Ergebnis als alle andere Teams: +0.2 Punkte (pro Team)
- Entdecken eines Bugs: 0.1 Punkte (pro Team)

Alle Untersuchungen sind schriftlich zu dokumentieren. Neben der schriftlichen Arbeit sind alle Quellen (Programmcode, Texte, Testbilder) und Tools abzugeben, damit eine Reproduktion der Ergebnisse möglich ist.

Teilaufgaben:

- Recherche
- Dokumentation (Grundlagen, Änderungen am Quellcode, Kompressionsergebnisse)

Max. 2 Personen, (maximal 3 Teams mit diesem Thema möglich)  
Max 1.5 Zusatzpunkte für Klausur

## **"Automatische Wahl von Parametern zur Kompression von Farbbildern"**

Eine vorhandene Software (TSIP) zur Kompression von Bilddaten analysiert das geladene Bild und wählt günstige Parameter zur Kompression aus. Die Auswahl ist jedoch nicht in jedem Fall optimal.

Auf Basis einer vorhandenen Datenbank, welche numerische Eigenschaften von Bildern und die besten Kompressionseinstellungen enthält, sind Zusammenhänge (Korrelationen) zu ermitteln

und die Frage zu beantworten, welche Einstellungen (automatisch) vorgenommen werden müssen, damit die komprimierte Datei möglichst klein ist.

Die herausgefundenen Zusammenhänge sind anhand vorhandener Testbilder zu prüfen.

Alle Untersuchungen sind schriftlich zu dokumentieren. Neben der schriftlichen Arbeit sind alle Quellen (Programmcode, Texte, Testbilder) und Tools abzugeben, damit eine Reproduktion der Ergebnisse möglich ist.

Teilaufgaben:

- Recherche
- Dokumentation (Grundlagen, Änderungen am Quellcode, Kompressionsergebnisse)

Max. 2 Personen, (maximal 3 Teams mit diesem Thema möglich)

Max 1.5 Zusatzpunkte für Klausur

## 5.2 ANALYSEERGEBNISSE DURCH SYSTEM MONITOR

## Auswertung FotoQuant

System Monitor Record	Starting date and time: AnotherMonitor (Pid 16452)	2016-01-27-22-37-32	Read interval (ms): FotoQuant (Pid 16923)	1000	MemTotal (kB)	1863064	Memory available		
Total CPU usage (%)	CPU usage (%)	AnotherMonitor Memory (kB)	CPU usage (%)	FotoQuant Memory (kB)	Memory used (kB)	(MemFree+Cached) (kB)	MemFree (kB)	Cached (kB)	Threshold (kB)
45.454.544	20.454.546	10300	0.0	20616	1023816	839248	225512	613736	225000
4.488.889	53.333.335	10300	0.0	20620	1008604	854460	240064	614396	225000
59.197.323	26.086.956	10300	0.33444816	20756	1015480	847584	232604	614980	225000
57.668.713	64.417.176	10300	18.404.907	98672	1091960	771104	153812	618160	225000
71.358.025	0.49382716	10300	43.209.877	79968	1082932	780132	162468	618284	225000
75.157.234	0.6289308	10300	55.345.913	94860	1104044	759020	140728	618292	225000
6.481.481	0.23148148	10300	4.722.222	108972	1115448	747616	128684	618312	225000
6.433.735	0.96385545	10300	4.939.759	102432	1104908	758156	140580	618320	225000
61.395.348	0.0	10300	4.465.116	82972	1088632	774432	156096	618336	225000
33.968.254	0.63492066	10300	10.793.651	85560	1089080	773984	154520	619464	225000
5.410.628	0.0	10300	28.502.415	84084	1087968	775096	153896	621448	225000
52.284.264	0.5076142	10300	3.654.822	88616	1092508	770556	152160	618396	225000
3.722.222	0.5555556	10300	10.555.555	11100	1115616	747448	126068	621380	225000
6.319.218	0.3257329	10300	40.716.614	102823	1107456	755608	134124	621484	225000
6.960.784	0.3267974	10300	43.790.848	110996	1116052	747012	125520	621492	225000
5.785.124	0.0	10300	3.829.201	102992	1108144	754920	133412	621508	225000
4.007.782	0.7782101	10300	11.284.047	116008	1118240	744824	124420	621520	225000
5.144.033	0.41152263	10300	4.074.074	127880	1129080	733984	110704	623280	225000
58.4	0.4	10300	32.4	91608	1098868	764196	140872	623324	225000
5.112.782	0.37593985	10300	37.369.925	88544	1092776	770288	150012	620276	225000
7.486.631	10.695.187	10300	14.438.502	83716	1114512	748552	124388	624164	225000
6.518.771	0.0	10300	36.177.475	87148	1116792	746272	122004	624268	225000
7.327.189	0.4608295	10300	4.746.544	90528	1120384	742680	118400	624280	225000
6.638.655	0.42016807	10300	42.016.808	91572	1121976	741088	116800	624288	225000
6.462.094	0.36101082	10300	37.545.128	91572	1122216	740848	116552	624296	225000
7.177.419	16.129.032	10300	37.903.225	91572	1122080	740984	116684	624300	225000
7.230.047	0.46948355	10300	46.948.357	97520	1126404	736660	112600	624308	225000
63.736.263	0.36630037	10300	39.194.138	95044	1125096	737968	113644	624324	225000
6.886.793	0.0	10300	41.981.133	98072	1125544	737520	113192	624328	225000
722.488	0.9569378	10300	4.880.383	99736	1127108	735956	111620	624336	225000
7.095.238	0.0	10300	46.190.475	99736	1127100	735964	111620	624344	225000
6.966.824	0.9478673	10300	45.023.697	99724	1127092	735972	111620	624352	225000
65.625	0.0	10300	421.875	83904	112080	750984	126624	624360	225000
63.779.526	0.39370078	10300	37.401.573	86988	1113564	749500	125140	624360	225000
68.348.625	0.4587156	10300	45.412.846	90240	1116648	746416	122040	624376	225000
7.339.449	0.4587156	10300	4.862.385	91576	1117884	745180	120800	624380	225000
71.875	0.0	10300	43.75	91576	1118372	744692	120304	624388	225000
689.243	0.3984064	10300	35.856.575	91576	1118700	744364	120016	624348	225000
7.214.612	0.456621	10300	48.401.825	111000	1137940	725124	100720	624404	225000
6.694.561	0.41841003	10300	41.004.185	94948	1122264	740800	116396	624404	225000
67.441.864	0.4651163	10300	4.744.186	98704	1125840	737224	112800	624424	225000
6.712.329	0.456621	10300	44.292.236	99740	1127568	735496	111064	624432	225000
71.818.184	0.90909094	10300	45.909.092	99740	1127568	735496	111064	624432	225000
6.869.159	0.93457943	10300	4.579.439	99728	1127436	735628	111180	624448	225000
6.124.402	0.9569378	10300	33.014.355	99740	1126048	737016	112560	624456	225000
40.070.923	0.35460994	10300	22.340.425	110716	1135156	727908	103436	624472	225000
60.747.665	0.46728972	10300	4.953.271	114728	1139392	723672	99192	624480	225000
54.545.456	0.4784689	10300	49.760.765	118416	1142732	720332	95844	624488	225000
53.271.027	0.46728972	10300	49.065.422	122125	1145824	717240	92744	624496	225000
56.108.597	0.4524887	10300	4.886.878	93180	1117548	745516	121016	624500	225000
55.140.186	0.46728972	10300	4.859.813	93180	1117664	745400	120892	624508	225000
5.479.452	0.0	10300	48.858.448	93180	1117656	745408	120892	624516	225000
5.504.587	0.9174312	10300	4.862.385	93180	1117772	745292	120768	624524	225000
543.379	0.0	10300	48.401.825	93180	1117900	745164	120644	624520	225000
55.707.764	0.913242	10300	48.858.448	93168	1118016	745048	120512	624536	225000
55.450.237	0.0	10300	45.971.565	125100	1150312	712752	86784	625968	225000
30.967.741	12.903.225	10300	6.451.613	125124	1149684	713380	90256	623124	225000
6.554.307	0.37453184	10300	39.325.844	91648	1127404	735660	109464	626196	225000
60.705.883	0.47058824	10300	4.517.647	96060	1121600	741464	115664	626296	225000
63.484.486	0.23866348	10300	48.926.014	96232	1122584	740480	114168	626312	225000
58.612.442	0.71770334	10300	45.933.014	80444	1116708	746356	122988	623368	225000
10.810.811	20.270.271	10300	0.6756757	80444	1105352	757712	134444	623268	225000
23.255.814	0.0	10300	0.7751938	80444	1105340	757724	134452	623272	225000
11.450.381	22.900.763	10300	0.0	80412	1105332	757732	134444	623288	225000
27.777.779	0.6944444	10300	0.6944444	80408	1102220	760844	137548	623296	225000
23.697.916	0.5208333	10300	0.5208333	49364	1058100	804964	181688	623276	225000
21.596.245	0.23474178	10300	0.0	49368	1046460	816604	193168	623436	225000
27.250.608	0.243309	10300	0.0	49368	1069784	793280	169720	623560	225000
4.207.317	0.6097561	10300	DEAD	DEAD	1018072	844992	221552	623440	225000

## Auswertung QuanPic

System Monitor Record	Starting date and time: AnotherMonitor (Pid 16452)	2016-01-27-22-41-51	Read interval (ms): QuanPic (Pid 19071)	1000	MemTotal (kB)	1863064	Memory available		
Total CPU usage (%)	CPU usage (%)	AnotherMonitor Memory (kB)	CPU usage (%)	QuanPic Memory (kB)	Memory used (kB)	(MemFree+Cached) (kB)	MemFree (kB)	Cached (kB)	Threshold (kB)
5.844.156	3.961.039	10300	0.0	6692	1068532	794532	173488	621044	225000
4.581.281	14.778.325	10300	0.0	6696	1057460	805604	184568	621036	225000
52.752.293	12.844.037	10300	0.9174312	7016	1059788	803276	182236	621040	225000
6.928.934	134.517.765	10300	22.335.026	24712	1092076	770988	148176	622812	225000
6.921.182	7.635.468	10300	3.546.798	24716	1092068	770996	148132	622864	225000
7.204.969	46.583.853	10300	4.006.211	24720	1091416	771648	148776	622872	225000
6.919.192	6.060.606	10300	33.838.383	30564	1100392	762672	138808	623864	225000
7.179.487	11.188.811	10300	33.799.534	30580	1100332	762732	138784	623948	225000
6.898.148	11.805.555	10300	33.101.852	30564	1099324	763740	139764	623976	225000
6.453.901	6.382.979	10300	38.061.466	30616	1099212	763852	139268	624088	225000
63.049.095	62.015.505	10300	30.232.557	36548	1106200	756864	132768	624096	225000
6.882.716	9.876.543</td								

## Auswertung QuantiPic

System Monitor Record	Starting date and time: AnotherMonitor (Pid 16452)	2016-01-27-22-40-31	Read interval (ms): QuantiPic (Pid 18467)	1000	MemTotal (kB)	1863064	Memory available			
Total CPU usage (%)	CPU usage (%)	Memory (kB)	CPU usage (%)	QuantiPic Memory (kB)	Memory used (kB)	(MemFree+Cached) (kB)	MemFree (kB)	Cached (kB)	Threshold (kB)	
20.0	58.333.335	10300	0.0	7960	1036624	826440	222360	604080	225000	
59.893.047	23.529.411	10300	0.0	7960	1012880	850184	245996	604188	225000	
54.924.244	15.909.091	10300	0.37878788	10484	989004	874060	269532	604660	225000	
61.944.443	36.111.112	10300	20.277.779	26232	1046676	816388	209556	606832	225000	
7.281.324	33.096.926	10300	3.617.021	26240	1042988	820076	213168	606908	225000	
75.433.525	63.583.813	10300	43.641.617	26244	1041988	821076	214164	606912	225000	
68.75	12.740.385	10300	36.778.847	26248	1042252	820812	213892	606920	225000	
71.052.635	10.047.847	10300	34.688.995	26504	1036392	826672	219392	607652	225000	
70.428.894	58.690.743	10300	35.214.447	26524	1040256	822808	214952	607856	225000	
63.636.364	62.937.064	10300	32.634.033	29632	1059844	803220	194804	608416	225000	
7.651.163	1.116.279	10300	36.976.746	33772	1062856	800208	191556	608652	225000	
7.203.792	11.611.375	10300	2.914.692	31272	1053980	809084	200008	609076	225000	
65.254.234	8.757.062	10300	31.355.932	31284	1047768	815296	206148	609148	225000	
7.165.109	43.613.706	10300	37.694.702	30324	1047644	815420	205896	609524	225000	
70.833.336	5.654.762	10300	3.422.619	31524	1048876	814188	204656	609532	225000	
6.556.604	12.264.151	10300	29.009.434	31908	1064688	798376	188836	609540	225000	
6.520.681	9.002.433	10300	3.381.995	32160	1065296	797768	188228	609540	225000	
6.892.308	58.461.537	10300	31.692.308	31792	1049016	814048	204492	609556	225000	
7.114.428	62.189.054	10300	36.069.653	60488	1078664	784400	174804	609596	225000	
6.797.066	11.735.941	10300	31.295.843	88940	1107084	755980	146352	609628	225000	
71.034.485	12.413.794	10300	34.252.872	120736	1150772	712292	102168	609628	225000	
6.682.243	77.102.804	10300	32.242.992	149024	1166204	696860	87344	609640	225000	
6.591.479	60.150.375	10300	3.508.772	46572	1089424	773640	163724	609916	225000	
37.453.182	8.988.764	10300	2.621.723	33820	1033156	829908	221768	608140	225000	
53.333.332	37.435.898	10300	0.0	33836	1033440	829624	221472	608152	225000	
47.031.963	31.050.228	10300	5.479.452	33928	1033356	829708	221540	608168	225000	
37.155.964	9.633.027	10300	50.458.717	33576	1033840	829224	221048	608176	225000	
5.357.143	8.928.572	10300	6.071.429	16564	1048040	815024	205136	609888	225000	
7.169.373	1.136.891	10300	30.858.469	62064	1076200	786864	176880	609984	225000	
63.805.103	10.904.872	10300	31.322.506	95228	1111000	752064	142048	610016	225000	
45.898.003	64.301.553	10300	62.084.255	86700	1122392	740672	132424	608248	225000	
3.753.425	9.863.013	10300	0.0	86704	1108632	754432	145912	608272	225000	
30.316.742	56.561.084	10300	0.0	86704	1129648	733416	125124	608292	225000	
42.611.683	21.993.128	10300	DEAD	DEAD	1037300	825764	217504	608260	225000	

## Auswertung QuantiPig

System Monitor Record	Starting date and time: AnotherMonitor (Pid 3772)	2016-02-01-23-30-29	Read interval (ms): QuantiPig (Pid 12926)	1000	1863064	Memory available			
Total CPU usage (%)	CPU usage (%)	Memory (kB)	CPU usage (%)	QuantiPig Memory (kB)	Memory used (kB)	(MemFree+Cached) (kB)	MemFree (kB)	Cached (kB)	Threshold (kB)
35.576.923	13.221.154	10376	0.0	7168	1098064	765000	198196	566804	225000
21.103.117	9.352.518	10376	0.0	7168	1087112	775952	209288	566664	225000
28.947.369	38.277.512	10376	0.23923445	7288	1063472	795952	232744	566788	225000
29.411.764	8.123.249	10376	11.204.482	6668	1085612	777452	210168	567408	225000
52.396.168	1.341.853	10376	0.9584665	6692	1082340	780724	213312	567412	225000
6.796.875	42.502	10376	359.375	26324	1117220	745844	176612	569232	225000
71.980.675	57.971.015	10376	44.927.536	26336	1114460	748604	179348	569256	225000
66.824.646	2.843.602	10376	4.620.853	26324	1114944	748120	178860	569260	225000
61.381.073	30.690.536	10376	3.299.233	26752	1109472	753592	180232	573360	225000
77.339.905	10.344.828	10376	39.901.478	27740	1116972	746092	171620	574472	225000
75.0	12.857.142	10376	40.714.287	30160	1108984	754080	179160	574920	225000
752.381	54.761.906	10376	42.857.143	30348	1108352	754712	179788	574924	225000
62.411.346	4.964.539	10376	37.588.654	37404	1106012	757052	181780	575272	225000
62.768.497	62.052.507	10376	34.606.205	38612	1107000	756064	180788	575276	225000
670.354	11.283.186	10376	34.513.275	34372	1112452	750612	175324	575288	225000
692.494	89.588.375	10376	37.288.136	37620	1104532	758532	183268	575264	225000
66.666.664	25.641.026	10376	38.694.637	42384	1109672	753392	178796	575464	225000
6.324.201	2.739.726	10376	34.931.507	34392	1117012	746052	171076	575472	225000
6.689.815	71.759.257	10376	35.416.668	32664	1100372	762692	187212	575480	225000
6.546.763	105.515.585	10376	34.532.375	39076	1119160	743904	168240	575664	225000
6.442.308	9.375	10376	37.740.383	38020	1107052	756012	180384	575628	225000
6.053.269	53.268.766	10376	33.171.913	37720	1108580	754484	178568	575916	225000
61.032.864	49.295.774	10376	37.089.203	36824	1104828	758236	182288	575948	225000
70.804.596	11.494.253	10376	33.793.102	34036	1106188	756876	180916	575960	225000
6.785.714	10.952.381	10376	36.190.475	37832	1105180	757884	181916	575968	225000
64.805.824	8.252.427	10376	3.567.961	37508	1105860	757204	181048	576528	225000
6.581.396	27.906.976	10376	3.744.186	39140	1121708	741356	164548	576808	225000
63.480.392	5.147.059	10376	3.627.451	39060	1107944	755120	178304	576816	225000
6.539.379	10.501.193	10376	35.322.197	32732	1101592	761472	184644	576828	225000
62.110.313	71.942.444	10376	3.764.988	33064	1101532	761532	183776	577756	225000
66.047.745	71.618.037	10376	35.809.017	35064	1110536	752528	172700	579828	225000
63.325.184	56.234.717	10376	38.875.305	34468	1117600	745464	165624	579840	225000
6.857.798	8.944.954	10376	35.321.102	38884	1115984	747080	167236	579844	225000
6.893.204	12.135.922	10376	34.951.458	39188	1117376	745688	165840	579848	225000
58.949.883	66.825.776	10376	3.508.353	39312	1111776	751288	171296	579868	225000
6.529.412	8.823.529	10376	35.294.117	39784	1108148	754916	175032	579884	225000
6.632.912	37.974.684	10376	33.670.887	36152	1108920	754144	171892	579896	225000
60.583.942	13.138.686	10376	27.980.536	53372	1121912	741152	160876	579904	225000
7.630.769	13.230.769	10376	39.076.923	58004	1128596	734468	152096	582496	225000
5.798.122	25.821.595	10376	29.342.724	53500	1124328	738736	156068	582668	225000
73.426.575	73.426.576	10376	3.811.189	59576	1132424	730640	148084	582680	225000
7.084.548	9.620.992	10376	36.734.695	67328	1133940	729124	145200	584048	225000
6.897.436	12.564.102	10376	3.102.564	44860	1115384	747680	163592	584092	225000
62.295.082	7.025.761	10376	30.679.157	35484	1117264	745800	161700	584100	225000
60.576.923	5.769.231	10376	29.086.538	45976	1114780	748284	164180	584104	225000
6.991.151	76.696.167	1037							