



Hochschule für Telekommunikation Leipzig
University of Applied Sciences

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

Studienmodul *ICT*
der Hochschule für Telekommunikation
Leipzig



Prüfungsvorleistung - Bilddatenkompression

vorgelegt von

Stefan Czogalla, Maik Lorenz, Jan Sutmöller, Stephan Kaden

24. Januar 2016

Dozent: Prof. Dr. habil. Tilo Strutz

INHALTSVERZEICHNIS

1 Einführung	2
1.1 Zweck	2
1.2 Produktanforderungen	2
1.3 Musskriterien	2
1.3.1 Kostenrahmen	2
1.4 Definitionen, Akronyme, Abkürzungen	2
2 Allgemeine Übersicht	3
2.1 Beschreibung der Ausgangssituation (Ist-Zustand)	3
2.1.1 Vergleich der vorhandenen drei APP's	3
2.2 Beispielbilder	6
2.3 Produkteinsatz	15
2.3.1 Anwendungsbereiche	15
2.3.2 Zielgruppen, Qualifikationsniveau	15
2.4 Produktfunktionalität	15
2.5 Randbedingungen	15
2.6 Annahmen und Abhängigkeiten	15
3 Anforderungen	15
3.1 Fachkonzept	15
3.1.1 Verwendete Bibliotheken von Drittanbieteren	15
3.1.2 betrachtete Quantisierungsverfahren	16
3.2 Anforderungen für Inbetriebnahme und Einsatz	17
3.2.1 Installationsprozedur	17
3.3 Qualitätsanforderungen	17
3.3.1 Qualitätsmerkmale	17
3.4 Anforderung an die Entwicklung	17
3.4.1 Entwicklungs-Umgebung	17
3.4.2 Änderungsmanagement	18
4 Ergebnis	18
4.1 Umfang der APP	18
4.2 Erscheinungsbild	20
4.3 Quellcode	22
4.3.1 Allgemein	22
4.3.2 MainActivity.java	22
4.3.3 CameraView.java	23
4.3.4 CameraListener.java	28
4.4 Beispielbilder	28
4.5 Bildgrößen	35
5 Anhang	35
5.1 PVL Aufgabenbeschreibung	36

1 EINFÜHRUNG

1.1 ZWECK

Dieses Dokument dient als Dokumentation des berufsbegleitenden Studienganges, Kommunikations- und Medieninformatik des Matrikel 13, mit der Programmierung einer APP zur Kompression von Bilddaten. Es setzt dabei die Rahmenbedingungen fest.

1.2 PRODUKTANFORDERUNGEN

Der Betrieb der Smartphone-APP muss auf allen gängigen Android-Smartphones ab Version 4.4.2 möglich sein.

Durch die APP wird den Studenten der HFTL ermöglicht:

- Fotos aufnehmen
- Bilder mit einer skalaren Quantisierung komprimieren
- Quantisierungsinterval soll einstellbar sein
- Ablage der komprimierten Bilder in passend zum Verfahren benannten Ordnern

1.3 MUSSKRITERIEN

Zunächst müssen zwingend folgende Punkte des Umfangs erfüllt werden:

- Bildaufnahme
- Quantisierung
- Auswahl Quantisierungsinterval
- Abspeichern mit passendem Dateinamen

1.3.1 KOSTENRAHMEN

Für die Entwicklung der APP soll auf kostenfreie Opensource-Programme oder auf vordefinierte Klassen für die Programmierung zurückgegriffen werden.

1.4 DEFINITIONEN, AKRONYME, ABKÜRZUNGEN

HfTL Hochschule für Telekommunikation Leipzig

APP Kurzform für Applikation

XML Extensible Markup Language

GUI Graphical User Interface

QuantiPig quantisiertes Picture

2 ALLGEMEINE ÜBERSICHT

2.1 BESCHREIBUNG DER AUSGANGSSITUATION (IST-ZUSTAND)

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

2.1.1 VERGLEICH DER VORHANDENEN DREI APP'S

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"> APP speichert Bilder je nach Verfahren in einem spezifischen Ordner setzt Zeitstempel bei Bildern 			<ul style="list-style-type: none"> hat OpenCV bereits integriert zeigt Auflösung des Bildes live an zeigt Livebild flüssig an (14fps) zeigt fps an 		<ul style="list-style-type: none"> Livebild bei Originalbild flüssig Livebild bei "Farbwerte" flüssig "Verarbeitete" Bilder sind kleiner als das Original(komprimiert) 		
Nachteile:	<ul style="list-style-type: none"> sehr langsame Verarbeitung in Echtzeit langsame Verarbeitung bei Originalbild verarbeitete Bilder sind größer als Orig. bei Midtread wird nur die obere Hälfte des Bildes quantisiert OpenCV muss nachträglich auf dem Smartphone installiert werden 			<ul style="list-style-type: none"> ruckelt stark bei "MedianBlur" (2fps) keine intuitive Bedienung 		<ul style="list-style-type: none"> OpenCV muss nachträglich auf dem Smartphone installiert werden Livebild bei "Helligkeit" stockend 		
Usability:	<ul style="list-style-type: none"> Quantisierung ist über Menü auswählbar besitzt einen Auslösebutton Hilfefunktion mit Beschreibung der einzelnen Menüpunkte 			<ul style="list-style-type: none"> keine Erklärung der Funktionen 		<ul style="list-style-type: none"> intuitive Bedienung Verfahren ist über ein Menü auswählbar 		

Tabelle 1: Vergleich der drei vorhandenen APP's

Vergleich Standard-APP zu QuantiPiq

	HTC	QuantiPiq					
Merkmale:	• Standard-APP	• drei Modi					
Modus:	Original	Original	Midtread	Skalar 2x2	Skalar 4x4	Skalar 8x8	Skalar 208x208
Dateityp:	JPG	JPG	JPG	JPG	JPG	JPG	JPG
Abmessung:	2688 x 1520	2688 x 1520	2688 x 1520	2688 x 1520	2689 x 1520	2690 x 1520	2691 x 1520
Dateigröße (KB):	953	1631	2433	1685	1822	1125	38
Kompression zum jeweiligen Original:	-	100%	-49%	-3%	-12%	31%	98%
Eigenschaften:	-	<ul style="list-style-type: none"> • Modus ist über Menü auswählbar • besitzt einen Auslösebutton • Quantisierung im Livebild • einstellbares skalares Intervall • speichert Bilder je nach Verfahren in einem spezifischen Ordner • setzt Zeitstempel bei Bildern 					

Tabelle 2: Vergleich Standard-APP zu QuantiPiq

2.2 BEISPIELBILDER



Abbildung 2.1: FotoQuant Originalbild



Abbildung 2.2: FotoQuant Lloyd-Modus



Abbildung 2.3: FotoQuant MidTread-Modus



Abbildung 2.4: Quanpic im Originalbild-Modus



Abbildung 2.5: Quanpic im MedianBlur-Modus



Abbildung 2.6: QuantiPic im Originalbild-Modus



Abbildung 2.7: QuantiPic im Helligkeits-Modus



Abbildung 2.8: QuantiPic im Farbwerts-Modus

2.3 PRODUKTEINSATZ

2.3.1 ANWENDUNGSBEREICHE

Aktuell soll die APP nur für die Studenten der HfTL zugänglich sein, welche ein Android-Smartphone besitzen.

2.3.2 ZIELGRUPPEN, QUALIFIKATIONSNIVEAU

Da bei der Nutzergruppe von Studenten mit Erfahrung im Umgang mit solchen APP's ausgegangen werden kann, wird auch die Oberfläche dementsprechend gestaltet.

2.4 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.“

2.5 RANDBEDINGUNGEN

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

Durch das Projektteam wird es nach Ende des Projektes keine weitere Softwarebetreuung, Wartung oder der gleichen geben. Es finden ebenfalls keine Schulungen oder Einweisungen statt.

2.6 ANNAHMEN UND ABHÄNGIGKEITEN

Die APP wird für Android-Geräte ab Version 4.0.3 zur Verfügung gestellt.

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.

3.1.1 VERWENDETE BIBLIOTHEKEN VON DRITTANBIETEREN

- OpenCV

3.1.2 BETRACHTETE QUANTISIERUNGSVERFAHREN

Ein digitales Bild ist immer nur eine Annäherung (Approximation) der Originalabbildung. Bei einem technischen bildverarbeitenden System wird (heute) fast ausschließlich ein kartesisches Basisgitter für das Bildraster benutzt. In der Biologie kommen diese Basisgitter oft, wie zum Beispiel die Photorezeptoren im Auge, in einer Wabenstruktur vor. Bei technischen Anwendungen wird ein hexagonales Gitter verwendet. Für die technische Bildverarbeitung werden rechteckige bzw. quadratische Strukturen verwendet, sogenannte Pixel. Somit wird eine einfache mathematische Behandlung über Matrizen ermöglicht.

Quantisierung

Unter Quantisierung versteht man die Bewertung der Helligkeit (Intensität) eines Pixels mittels einer festgelegten Grauwert- bzw. Farben-Menge, z.B. natürliche Zahlen von 0 bis 255.

Skalare Quantisierung

Die skalare Quantisierung ordnet jedem Eingangswert einen quantisierten Wert aus einer endlichen Wertemenge zu. Die Zuordnung erfolgt dabei im einfachsten Fall linear auf Basis eines Rasters mit Intervallen fester Länge. Durch die Abbildung aller Eingangswerte innerhalb eines bestimmten Intervalls auf denselben quantisierten Wert entsteht verlustbehaftete Datenkompression. Um bestimmte Werte stärker zu quantisieren als andere, können anstelle eines festen Rasters auch unterschiedliche Intervallbreiten gewählt werden. Durch die Einschränkungen der menschlichen Wahrnehmung kann es beispielsweise Sinn machen unterschiedliche Intervallbreiten zu verwenden. Dieses Verfahren wird als nichtlineare Quantisierung bezeichnet.

Vektorquantisierung

Die Vektorquantisierung berücksichtigt mehrere Signalwerte gleichzeitig, die als Vektor des mehrdimensionalen Raums aufgefasst werden. Wie die skalare Quantisierung stellt auch die Vektorquantisierung einen verlustbehafteten Vorgang dar. Ein Vektorquantisierer bildet Eingabevektoren auf eine endliche Menge ab, die aus Ausgabevektoren besteht. Für die Wahl der Ausgabevektoren können verschiedene Kriterien herangezogen werden. Im einfachsten Fall kommt das euklidische Abstandsmaß der Vektoren zum Einsatz. Die Menge 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.

verwendetes Verfahren

Auf Grund der einfacheren Implementierung haben wir uns für die skalare Quantisierung

entschieden,

Skalare Quantisierung	Vektorquantisierung
Eingangswert $x \in \mathbb{R}$	Eingangsvektor $x \in \mathbb{R}^M$
Ausgangswert $y \in \mathbb{R}$	Ausgangsvektor $y \in \mathbb{R}^M$
Mögliche Ausgangswerte y_1, \dots, y_N	Codevektoren $y_1, \dots, y_N \in \mathbb{R}^M$
Quantisierungsintervall	Quantisierungszelle

Abbildung 3.1: Quantisierungsverfahren

3.2 ANFORDEUNGEN FÜR INBETRIEBNAHME UND EINSATZ

3.2.1 INSTALLATIONSPROZEDUR

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

3.3 QUALITÄTSANFORDERUNGEN

3.3.1 QUALITÄTSMERKMALE

Folgende Qualitätsansprüche werden gestellt:

- Hohe Zuverlässigkeit der Software
- schnelle und zuverlässige Verarbeitung der gewünschten Daten
- Fehler werden mit einer entsprechenden Fehlermeldung beantwortet
- Intuitiv benutzbar
- Leicht zu warten und zu erweitern
- Vollständige Dokumentation des Projektes

3.4 ANFORDERUNG AN DIE ENTWICKLUNG

3.4.1 ENTWICKLUNGS-UMGEBUNG

Für die Entwicklung wird Android Studio inkl. Gradle in der Version 1.x genutzt. Für die Dokumentation und Projektkoordination wird GitHub verwendet. Die Dokumentation wird mittels L^AT_EX erstellt.

3.4.2 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 UMFANG DER APP

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



Abbildung 4.1: QuantiPig Logo

Als Algorithmen wurde folgende Verfahren implementiert:

- keine Quantisierung
- skalare Quantisierung mit folgenden Quantisierungsintervallen
 - 2x2 px
 - 4x4 px
 - 8x8 px
 - größtmögliche Quadrate (ggT)

- Midtread-Quantisierung

4.2 ERSCHEINUNGSBILD

Die APP startet generell im Landscape-Modus. Wie in der folgenden Abbildung zu sehen, gibt es jeweils einen Button für die Wahl des Quantisierungsverfahrens und den Auslöser.



Abbildung 4.2: QuantiPig Startbildschirm

Nach Drücken auf den Auslöser wird ein Foto mit aktuellem Komprimierungsverfahren 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 folgender Abbildung zu sehen, ein Auswahlmenü für die drei verschiedenen Verfahren.

Im Modus skalare Quantisierung gibt es einen weiteren Button mit dem man das Quantisierungsintervall auswählen kann.

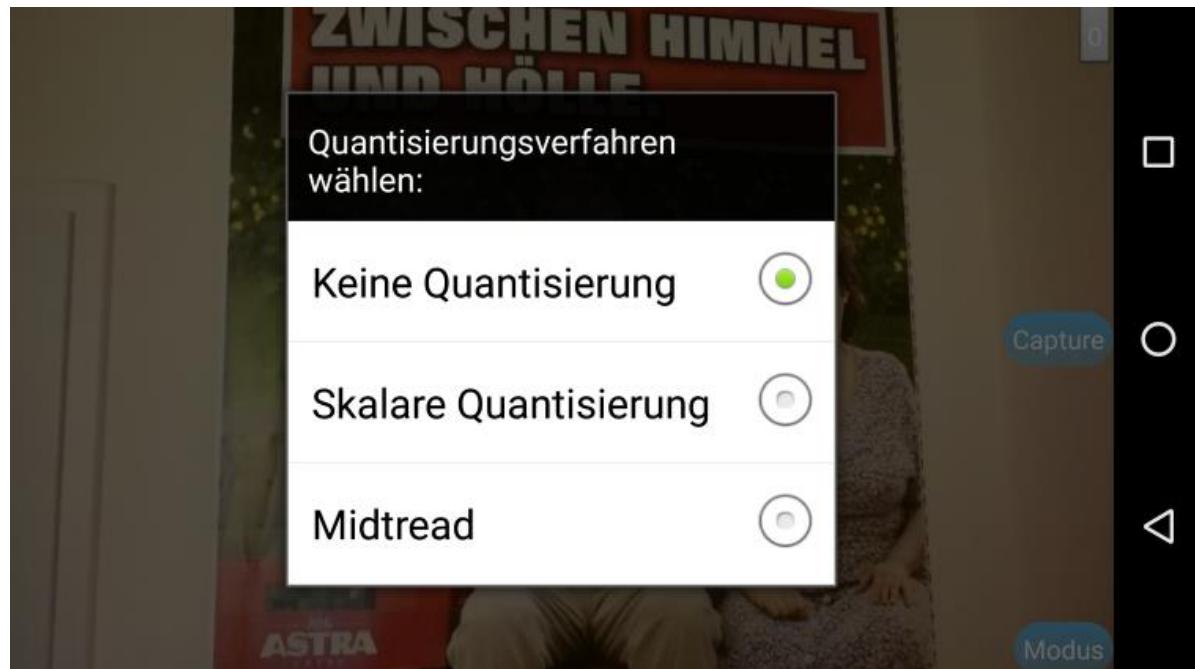


Abbildung 4.3: QuantiPig-Menü Verarbeitungsverfahren

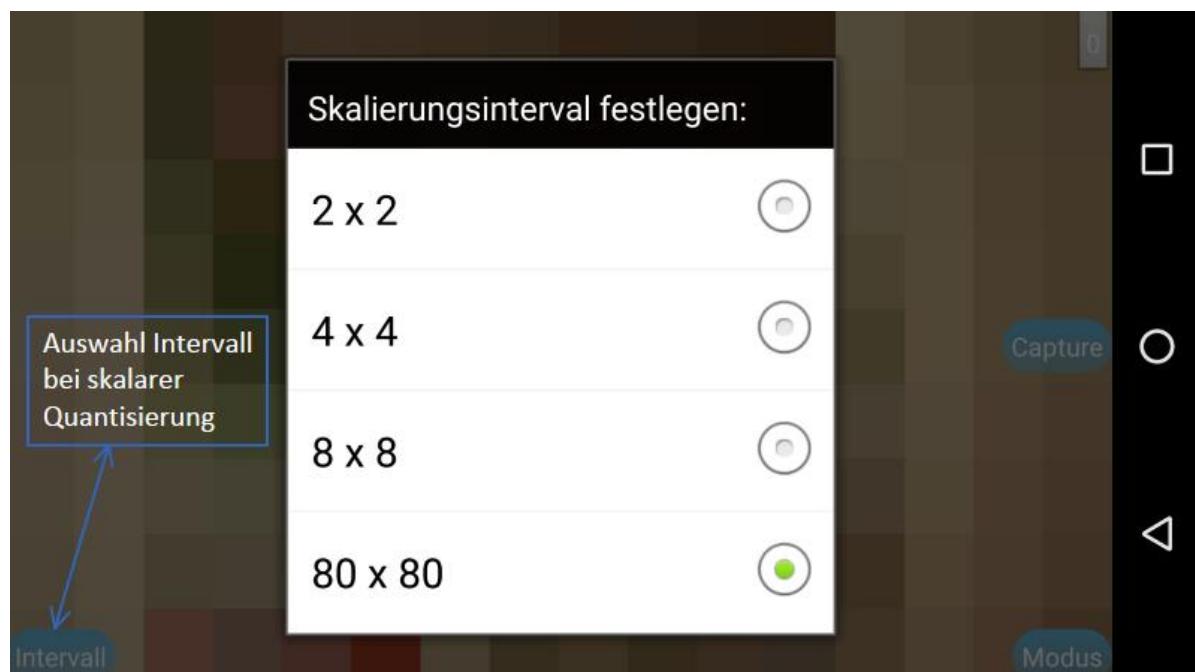


Abbildung 4.4: QuantiPig-Menü Quantisierungsintervall

4.3 QUELLCODE

4.3.1 ALLGEMEIN

Der grundlegende Aufbau des Quellcodes wurde von der vorgegebenen App “FotoQuuant“ übernommen. So wurden ebenfalls drei Klassen generiert:

- MainActivity.java
- CameraView.java
- CameraListener.java

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

4.3.2 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(R.layout.activity_main);

mCameraView = (CameraView) findViewById(R.id.surface_view);
mCameraView.setVisibility(SurfaceView.VISIBLE);
mCameraView.setCvCameraViewListener(new CameraListener());
```

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 (hier wird die Klasse **CameraView** aufgerufen)
- onPause()
- onDestroy();
- onStart();

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

4.3.3 CAMERAVIEW.JAVA

Dies ist das funktionale Kernstück der App. Hier werden die Bildbearbeitungsalgorithmen auf das Live-Bild und auf das zu speichernde Bild gelegt.

quantiMode0 - Original

Dieser Modus ist der Standardmodus beim Starten der App und kann als Standardrahmen für die anderen Modi gesehen werden.

Diese Methode dient als reiner Durchlauf und soll lediglich sicherstellen, dass das *unbearbeitete* Bild genauso behandelt wird, wie die *bearbeiteten* Bilder. Der Methode wird das Bild als Array mitsamt Höhe und Breite übergeben und die Methode gibt das Array unbehandelt wieder zurück.

```
public static byte[] quantiMode0(byte[] buff, int mHeight, int
    mWidth) {
    return buff;
}
```

quantiMode1 - Skalar

Dieser Modus ist der aufwändigste. Allgemein wird nichts anderes gemacht, als die RGBA-Werte der Bildpunkte innerhalb eines Quadrates (die Kantenlänge entspricht dem **Intervall**) in ein neues Array (kanalweise) addiert. Der Index dieses neuen Arrays entspricht dabei dem abgetasteten Intervallquadrat. Sobald dies mit dem kompletten Bild geschehen ist, wird der Durchschnitt aus diesen RGBA-Werten (kanalweise) errechnet und wieder zurück in das originale Array übergeben. Somit entsteht - je nach Größe des Intervalls - ein immer mehr grobkörniges Bild.

Die erste Besonderheit dieses Algorithmus ist, dass dieser nicht direkt durch Auswahl im Menü aufgerufen wird, sondern über den Umweg über die Methode setCluster(). Dies führt zur zweiten Besonderheit: Diese Methode bekommt neben dem Array und den Werten für Breite und Höhe des Bildes zusätzlich noch das Intervall (*cluster*) als Parameter übergeben.

```
public static byte[] quantiMode1(byte[] buff, int mHeight, int
    mWidth, int cluster){
    ...
    return 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
    * 4];
```

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 Hfsarray 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 / cluster; m++) {
```

Danach wird jeder Bildpunkt im Cluster durchlaufen.

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

Und nun werden die RGBA-Kanäle des Bildpunktes durchlaufen.

```
for (int i = 0; i < 4; 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.



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

```

        if (buff[x + i + j * 4] >= 0)
            frame[k + i] += ((int) buff[x + i + j
                * 4]);
        else
            frame[k + i] += ((int) buff[x + i + j
                * 4] & 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 gefüllt. 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 += 4;
x += 4 * cluster;
}

```

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 < cluster)
    k -= mWidth * 4 / cluster;
```

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 Ganz Zahldivision stattfindet, was am Ende zu Rundungsfehlern führt.

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

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

quantiMode2 - Midtread

Dieser Modus wurde aus der App FotoQuant übernommen und optimiert. Dabei wird der jeweilige Kanalwert bitweise um 5 Stellen nach rechts verschoben, so dass die 5 niederwertigsten Bits aus dem Array entfernt werden. Die 5 höchstwertigen Bit werden - je nach Vorzeichen des Wertes mit 1 (bei negativen Werten) oder 0 (bei positiven Werten) aufgefüllt.

Anschließend wird der Inhalt des Arrays wieder bitweise um 5 Stellen nach links verschoben. Somit werden die 5 niederwertigsten Bit auf 0 gesetzt.

```
buff[t + j] = (byte) ((buff[t + j] >> 5) << 5);
```

Dadurch verringert sich der Farbraum von 16777216 (256 x 256 x 256) auf 512 (8 x 8 x 8) Farben. (*Der Alphakanal wurde in der Berechnung nicht berücksichtigt.*)

Die Optimierung des Algorithmus bestand zum Einen aus der Entfernung einer if-Abfrage, ob die Werte im Arrays größer oder kleiner als 0 sind und zum Anderen werden alle Bildpunkte des Live-Bildes abgetastet und bearbeitet (vgl. [Midtread\(FotoQuant\)](#)

setCluster

Folgende Intervalle sind implementiert:

- 2 x 2 (Standard)
- 4 x 4
- 8 x 8
- (Max) x (Max)

Im **Modus 8 x 8** müsste grundlegend eine Prüfung stattfinden, ob das Display restlos in Quadrate der Größe 8 aufteilbar ist, da sonst der Algorithmus in eine OutOfBoundsException läuft und die App anschließend abstürzt. Da dies jedoch nicht für die *gängigen* Auflösungen zutrifft, wurde auf diese Sicherheitsmaßnahme der Einfachheit halber verzichtet.

Der letzte Modus kann als eine experimentelle Spielerei angesehen werden. Hier wird über die Methode `ggT` die maximale Intervallbreite des Displays ermittelt und angewandt. So werden bei einer Auflösung von 1920 x 1080 ein Bild von 16 x 9 Quadra-ten mit der Kantenlänge von 120 Bildpunkten angezeigt.

ggT

Die Methode errechnet aus den übergebenen Werten (Höhe und Breite des Bildes) den größten gemeinsamen Teiler nach dem Euklidischen Algorithmus und übergibt diese an das **letzte Intervall** im **Skalar-Modus**.

```
private static int ggT(int zahl1, int zahl2){
    while (zahl2 != 0) {
        if (zahl1 > zahl2) {
            zahl1 = zahl1 - zahl2;
```

```
        }
        else {
            zahl12 = zahl2 - zahl1;
        }
    }
    return zahl1;
}
```

4.3.4 CAMERALISTENER.JAVA

Diese Klasse nutzt als Interface die OpenCV-Klasse

“*CameraBridgeViewBase.CvCameraViewListener2*“

Die Methode “*onCameraFrame()*“ erhält Einzelbilder von der Kamera und übergibt dieses an die Open-CV Klasse “*Mat*“ als RGBA-Matrix, welches wiederum von der Methode übergeben wird.

```
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame
    inputFrame) {
    Mat mRgba = inputFrame.rgba();
    return mRgba;
}
```

s. hierzu OpenCV-Dokumentation:

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

4.4 BEISPIELBILDER



Abbildung 4.6: QuantiPig Originalbild



Abbildung 4.7: QuantiPig Midtread



Abbildung 4.8: QuantiPig skalare Quantisierung 2x2 px



Abbildung 4.9: QuantiPig skalare Quantisierung 4x4 px



Abbildung 4.10: QuantiPig skalare Quantisierung 8x8 px

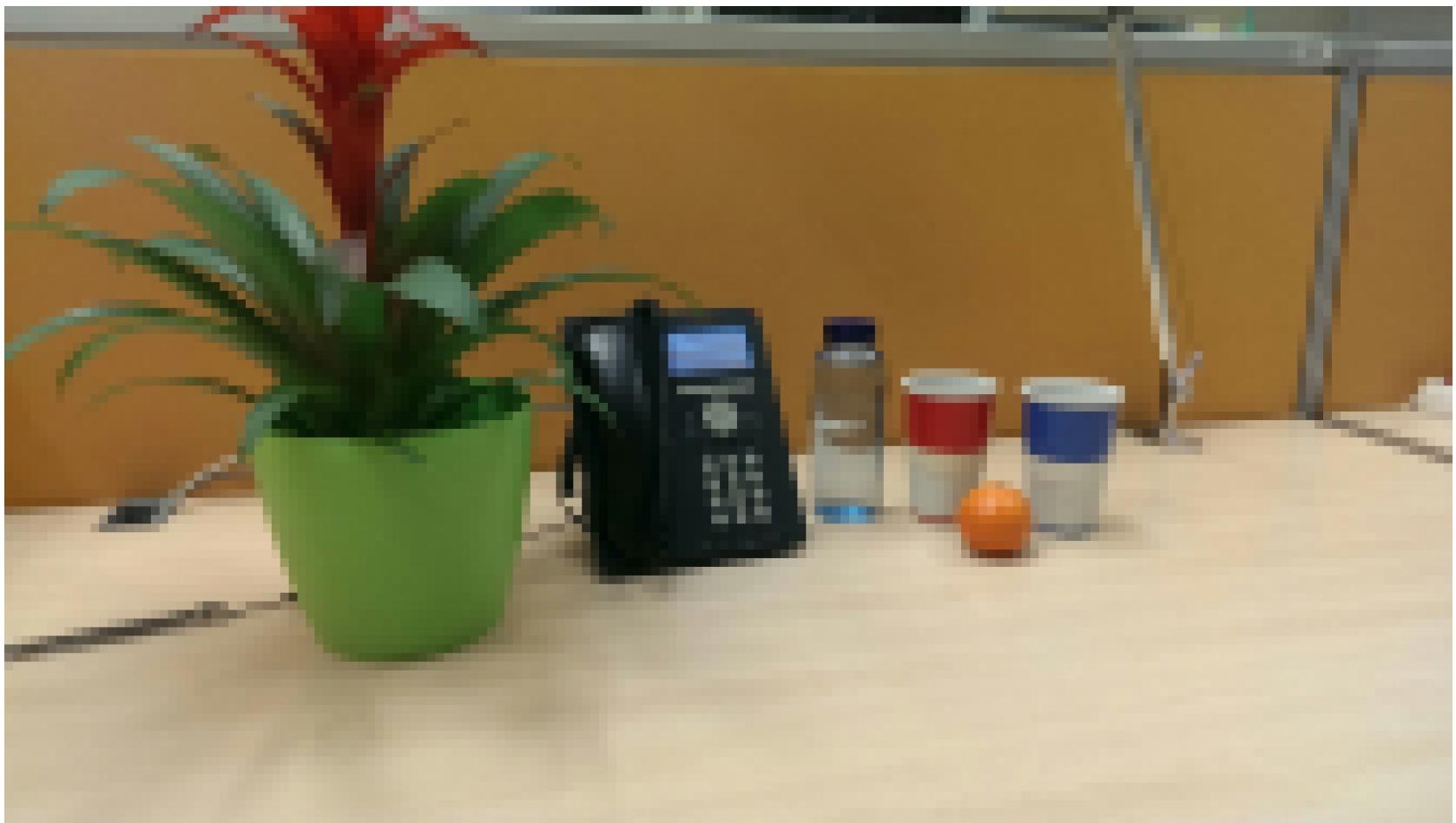


Abbildung 4.11: QuantiPig skalare Quantisierung 80x80 px

4.5 BILDGRÖSSEN

- Original: 3,8 MB
- skalare Quantisierung:
 - 2x2: 3,8 MB
 - 4x4: 2,6 MB
 - 8x8: 0,58 MB
 - 80x80: 0,12 MB
- Midtread: 4,9 MB

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