



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**

1. Februar 2016

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

# INHALTSVERZEICHNIS

<b>1 Einführung</b>	<b>3</b>
1.1 Zweck . . . . .	3
1.2 Produktanforderungen . . . . .	3
1.3 Musskriterien . . . . .	3
1.3.1 Kostenrahmen . . . . .	3
1.4 Definitionen, Akronyme, Abkürzungen . . . . .	3
<b>2 Allgemeine Übersicht</b>	<b>4</b>
2.1 Beschreibung der Ausgangssituation (Ist-Zustand) . . . . .	4
2.1.1 Vergleich der vorhandenen drei APP's . . . . .	4
2.2 Beispielbilder . . . . .	7
2.3 Produkteinsatz . . . . .	16
2.3.1 Anwendungsbereiche . . . . .	16
2.3.2 Zielgruppen, Qualifikationsniveau . . . . .	16
2.4 Produktfunktionalität . . . . .	16
2.5 Randbedingungen . . . . .	16
2.6 Annahmen und Abhängigkeiten . . . . .	16
<b>3 Anforderungen</b>	<b>16</b>
3.1 Fachkonzept . . . . .	16
3.1.1 Verwendete Bibliotheken von Drittanbietern . . . . .	16
3.1.2 betrachtete Quantisierungsverfahren . . . . .	17
3.2 Anforderungen für Inbetriebnahme und Einsatz . . . . .	18
3.2.1 Installationsprozedur . . . . .	18
3.3 Qualitätsanforderungen . . . . .	18
3.3.1 Qualitätsmerkmale . . . . .	18
3.4 Anforderung an die Entwicklung . . . . .	19
3.4.1 Entwicklungs-Umgebung . . . . .	19
3.4.2 Änderungsmanagement . . . . .	19
<b>4 Ergebnis</b>	<b>19</b>
4.1 Übernommene Funktionalitäten . . . . .	19
4.2 Nicht übernommene Funktionalitäten . . . . .	20
4.3 Eigene Entwicklungen . . . . .	20
4.4 Umfang der APP . . . . .	21
4.5 Erscheinungsbild . . . . .	22
4.6 Quellcode . . . . .	24
4.6.1 Allgemein . . . . .	24
4.6.2 MainActivity.java . . . . .	24
4.6.3 Pixel.java . . . . .	28
4.6.4 Skalar.java . . . . .	32
4.7 Beispielbilder . . . . .	34

4.8 Bildgrößen . . . . .	41
<b>5 Anhang</b>	<b>41</b>
5.1 PVL Aufgabenbeschreibung . . . . .	42

# 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"> <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 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"> <li>• Modus ist über Menü auswählbar</li> <li>• besitzt einen Auslösebutton</li> <li>• Quantisierung im Livebild</li> <li>• einstellbares skalares Intervall</li> <li>• speichert Bilder je nach Verfahren in einem spezifischen Ordner</li> <li>• setzt Zeitstempel bei Bildern</li> </ul>					

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

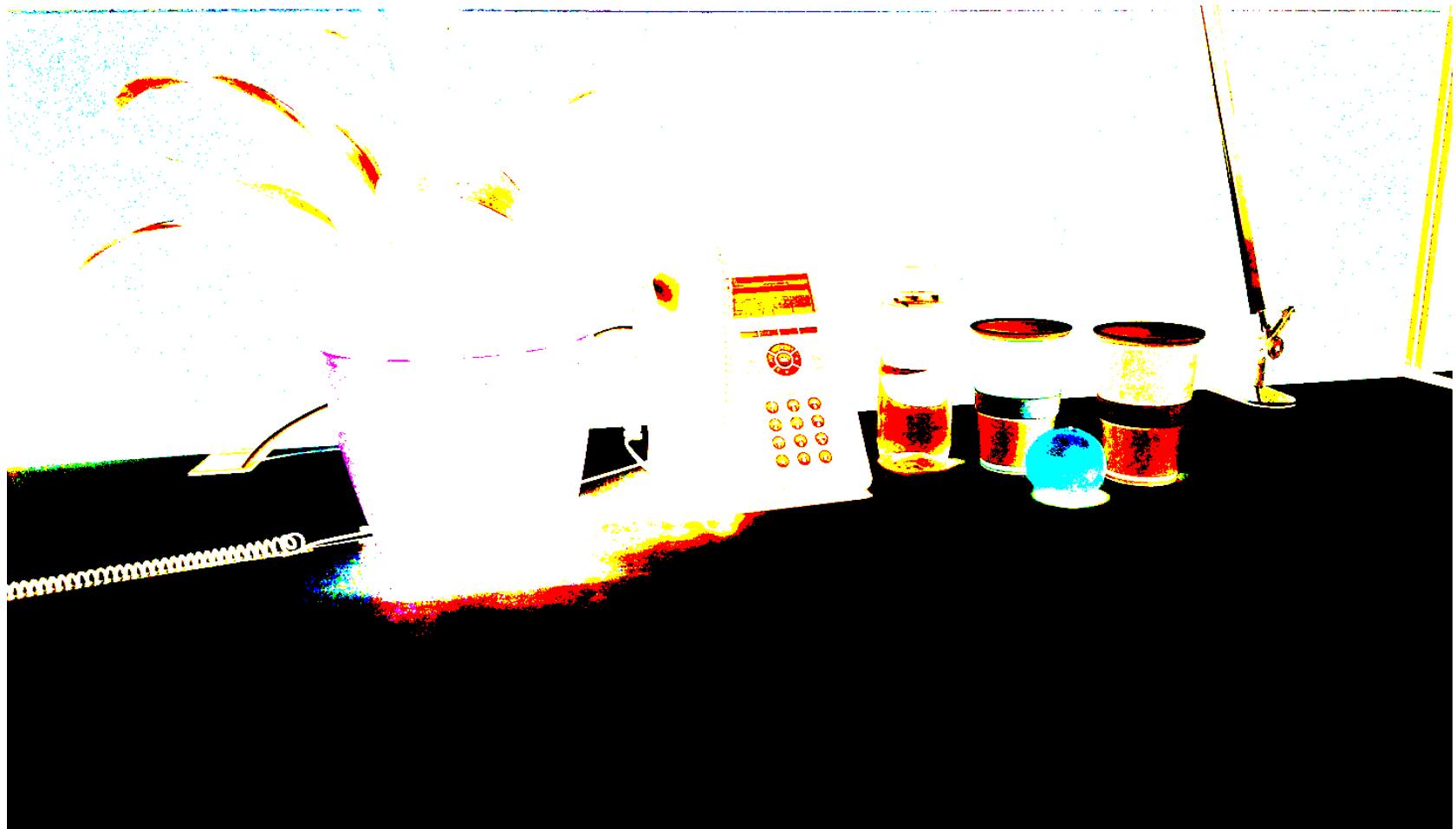


Abbildung 2.8: QuantiPic im Farbwerts-Modus

## 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 kommt diese Basisgitter oft, wie zum Beispiel die Photorezeptoren im Auge, in einer Wabenstruktur vor. Bei technischen Anwendungen wird ein hexagonales Gitter verwendet.[3] 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. [1]

#### **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,

<b>Skalare Quantisierung</b>	<b>Vektorquantisierung</b>
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: [2] Quantisierungsverfahren

- [1] 'Kompendium Medieninformatik: Mediennetze, Roland Schmitz, Roland Kiefer, Johannes Maucher, Jan Schulze, Thomas Suchy, Springer-Verlag, 2006'
- [2] 'Signaltheorie und Kodierung, Peter Vogel, Springer-Verlag, 2013'
- [3] '(<http://www.golem.de/news/facettenaugen-forscher-entwickeln-insektenauge-fuer-drohnen-1508-115588.html>)'

## 3.2 ANFORDEUNGEN FÜR INBETRIEBNAHME UND EINSATZ

### 3.2.1 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 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<sup>A</sup>T<sub>E</sub>X 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 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 “QuantiPig“ 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
  - Vorgerfertigte 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

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
- 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 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 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 folgender Abbildung zu sehen, ein Auswahlmenü für die drei verschiedenen Verfahren.

Im Modus “Pixel“ und “Skalar“ gibt es einen weiteren Button mit dem man das Intervall auswählen kann.

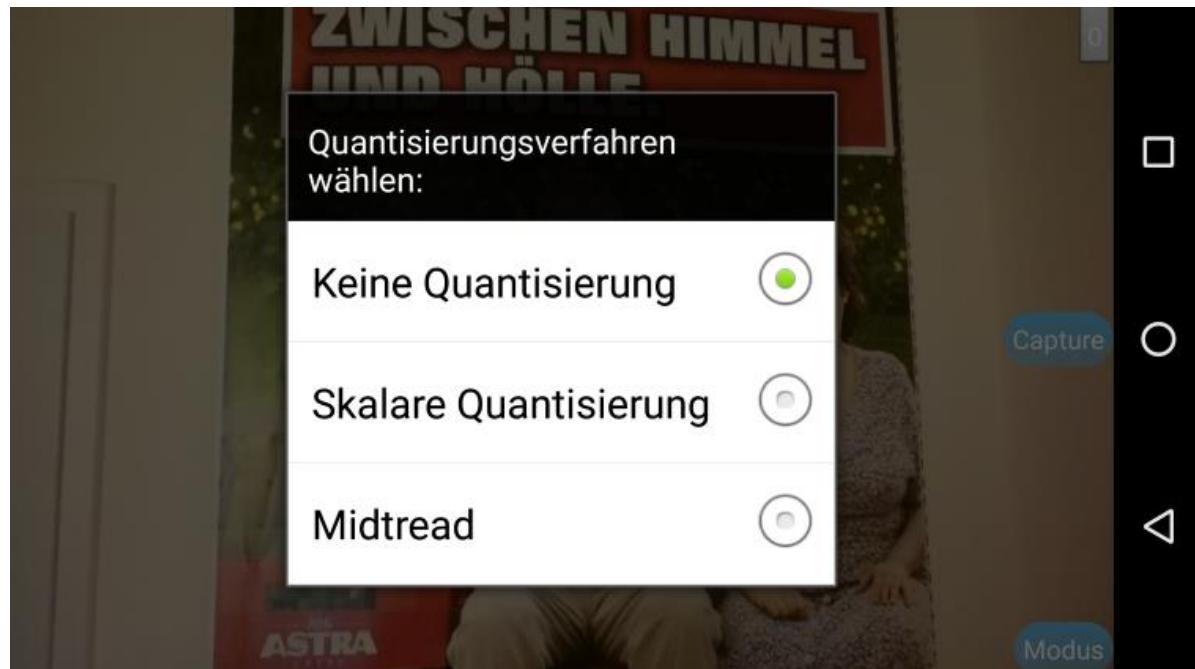


Abbildung 4.3: QuantiPig-Menü Verarbeitungsverfahren

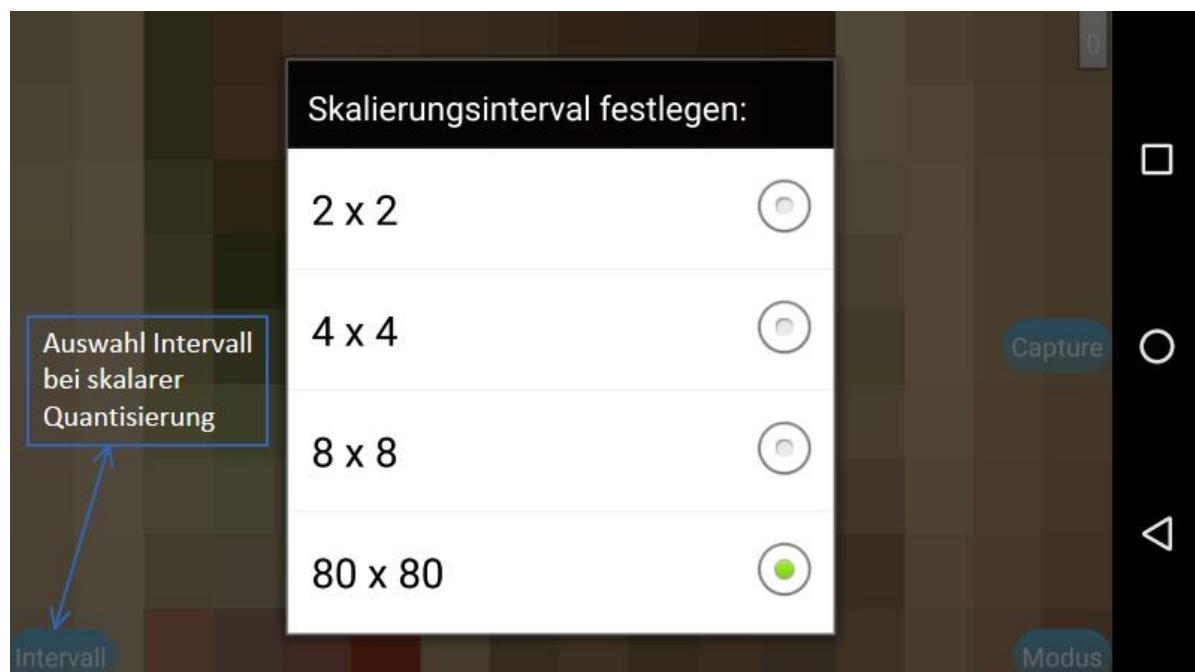


Abbildung 4.4: QuantiPig-Menü Quantisierungsintervall

## 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 `int` (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 Bit 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 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

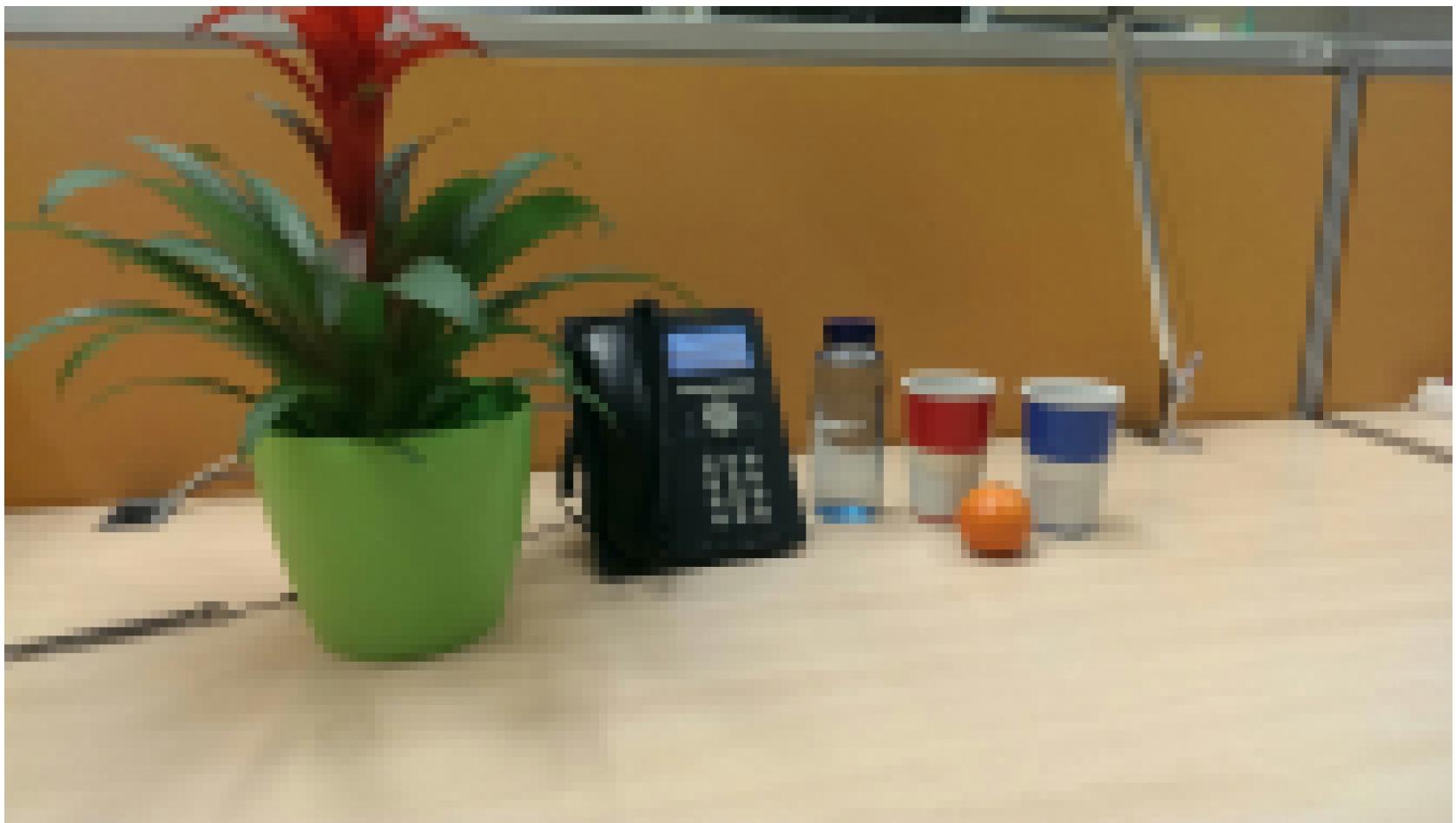


Abbildung 4.11: QuantiPig skalare Quantisierung 80x80 px

## 4.8 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