

Studienarbeit

Entwicklung einer Applikation zur Erkennung und Bearbeitung fehlerhafter Bilder

Studiengang Informatik

Cassandra Schilling Matrikel-Nr. 5919209

Elvira Kraft Matrikel-Nr. 3653393

Bearbeitungszeitraum: 300 Stunden pro Studierende

Abgabe 10.06.2022

Betreut von M.Sc. Jonas Fritzsch

Zusammenfassung

Bildschäden können durch verschiedenste Störfaktoren verursacht werden. Im Rahmen dieser Arbeit wird über Bilder gesprochen, die von geschädigten Datenträgern kommen. Die meisten solcher Bilder enthalten schadensbedingt ausgegraute oder bunte Bereiche, die an beliebigen Randpositionen entstehen können. Ziel dieser Arbeit ist es, eine Anwendung für die Reparatur der genannten Bildschäden zu erstellen. Hierzu wurden verschiedene Programmiersprachen, Bibliotheken und Algorithmen für die Bildverarbeitung untersucht, und es wurde festgestellt, dass die Programmiersprache Python und die Bibliothek OpenCV aufgrund der großen spezialisierten Funktionalität am besten geeignet sind. Nach der umfassenden Recherche und Analyse wurde eine Applikation in Python entwickelt, die unterschiedlich positionierte, geschädigte Bildbereiche erkennen und das Bild entsprechend zuschneiden kann. Die Anwendung erkennt von den 4 vorgestellten Schadensarten erst 2, wovon eine laut eigener Statistik besonders häufig vorkommt. Dadurch kann die Anwendung bereits erfolgreich bei Kundenaufträgen eingesetzt werden.

Abstract

Image damage can be caused by various factors. In the context of this work, we will talk about the images that come from damaged media. Most of those images contain grey or coloured areas caused by damage, that can appear at any position of the image. Various programming languages, libraries and algorithms for the image processing have been analysed, and it was found that the Python programming language and the OpenCV library are the most suitable for creating an application for damage repair on images. After the extensive research and analysis, an application was developed, which can detect differently positioned damaged areas of the image and crop it. Of the 4 types of damage presented, the application recognizes only 2, one of which, according to our statistics, occurs particularly frequently. Therefore the application can already be used successfully for customer orders.

Ehrenwortliche Erklärung

Wir versichern hiermit, dass wir die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Falls sowohl eine gedruckte als auch elektronische Fassung abgegeben wurde, versichern wir zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 10.06.2022

Kraft

Ort, Datum

Unterschrift E. Kraft

Cassandra Schilling

Unterschrift C. Schilling

Inhaltsverzeichnis

Abkürzungs- und Symbolverzeichnis	vii
Abbildungsverzeichnis	viii
Tabellenverzeichnis	ix
1 Einleitung	1
1.1 Motivation	1
1.2 Ziel der Arbeit	1
1.3 Gliederung der Arbeit	1
2 Aufgabenstellung	2
2.1 Typische Bildfehler und deren Ursache	2
2.2 Aufgabenstellung	4
3 Bildverarbeitung	5
3.1 Bilder im digitalen Kontext	5
3.2 Farträume und Farbmodelle	6
3.2.1 Rot-Grün-Blau (RGB)	6
3.2.2 Hue Saturation Value(Hue Saturation Value (HSV))	6
3.3 Fehler in Joint Photographic Experts Group (JPEG) Bildern	7
3.4 Ansätze für die Bildreparatur	8
3.5 Ablauf der Bildverarbeitung	8
3.6 Zusammenfassung	9
4 Verwandte Arbeiten	10
4.1 Fachliteratur	10
4.2 Wissenschaftliche Arbeiten	10
5 Analyse der Bibliotheken	12
5.1 JavaScript	12
5.1.1 sharp.js	12
5.1.2 CamanJS	13
5.2 Python	14
5.3 C++	16
5.4 Wahl der Sprache und Bibliothek	17
6 Analyse der Algorithmen	20
6.1 Algorithmen zum Einlesen mehrerer Bilder	20
6.2 Algorithmen für Merkmalerkennung (Feature Detection)	20
6.2.1 Segmentierungsalgorithmen	21
6.2.1.1 Wasserscheidentransformation (Watershed)	22
6.2.1.2 Mean-Shift-Clustering	23
6.2.1.3 Schwellenwertverfahren (Thresholding)	24

6.2.1.4	Canny Algorithmus	27
6.2.1.5	Laplace Filter	28
6.2.2	Algorithmen für die Texturenanalyse	30
6.2.2.1	Binary Gabor Pattern	31
6.2.2.2	Morphologische Filter	32
6.2.2.3	Energievariation	34
6.2.2.4	Eigenfilter	35
6.3	Algorithmen für Bildreparatur (Image Restoration)	36
6.3.1	Inverse Filterung	36
6.3.2	Wiener Filterung	37
6.3.3	Methode der kleinsten Quadrate	38
6.3.4	Gauß-Filter	39
6.4	Zusammenfassung	40
7	Anwendung für die Bildreparatur (ImageRepair)	41
7.1	Einführung und Anforderungen	41
7.1.1	Funktionale Anforderungen	41
7.1.1.1	Bildverarbeitung	41
7.1.1.2	Benutzeroberfläche	41
7.1.2	Nichtfunktionale Anforderungen	42
7.2	Ablauf der Bildverarbeitung	42
7.3	Implementierung der Anwendung	42
7.3.1	Benutzeroberfläche	43
7.3.2	Hauptprogramm	44
7.4	Zusammenfassung	44
8	Evaluierung der Ergebnisse und Reflexion	47
8.1	Testergebnisse und Bewertung	47
8.2	Reflexion	48
9	Zusammenfassung und Ausblick	49
Literaturverzeichnis		51

Abkürzungsverzeichnis

AVIF AV1 Image File Format

BGP Binary Gabor Pattern

BRIEF Binary Robust Independent Elementary Features

CLSF Constrained-Least-Squares-Filter

CMY Cyan-Magenta-Yellow

CMYK Cyan-Magenta-Yellow-Key

CSV Comma-Separated Values

DICOM Digital Imaging and Communications in Medicine

EoI End of Image

FITS Flexible Image Transport System

FLANN Fast Library for Approximate Nearest Neighbors

GIF Graphics Interchange Format

GUI Graphical User Interface

HDR High Dynamic Range

HEIC High Efficiency Image File Format

HSI Hue Saturation Intensity

HSV Hue Saturation Value

ICC International Color Consortium

JPEG Joint Photographic Experts Group

LBP Local Binary Pattern

LoG Laplacian of Gaussian

MCU Minimum Coded Unit

MLL Machine Learning Library

NIfTI Neuroimaging Informatics Technology Initiative

PDF Portable Document Format

PFM Portable FloatMap

PGM Portable Graymap

PNG Portable Network Graphics

PPM Portable Pixmap

PSF Point Spread Function

RAM Random-Access-Memory

RGB Rot-Grün-Blau

ROI Region Of Interest

SD-Karte Secure Digital Memory Karte

SIFT Scale-invariant feature transform

SNR Signal-Noise Ratio

SoI Start of Image

SoS Start of Scan

SURF Speeded-Up Robust Features

SVG Scalable Vector Graphics

TIFF Tagged Image File Format

XML Extensible Markup Language

YAML Yet Another Markup Language

Abbildungsverzeichnis

2.1	Bild unterteilt in Segmente	3
2.2	Bild enthält einen Graubereich	3
2.3	Bildteil weist andere Farbe auf	3
2.4	Bild enthält einen gestreiften Bereich	3
3.1	Aufbau eines Farbbildes in seinen Pixelarrays	5
3.2	RGB Farbraum	6
3.3	HSV Farbraum	7
3.4	Winkel zur Bestimmung des Farbtone	7
3.5	Aufbau des JPEG Formats	8
3.6	Ablauf der Bildverarbeitungsschritte	8
5.1	dtype-Bereiche	15
6.1	Originalbild	21
6.2	Bild nach Segmentierung	21
6.3	Illustration einer Wasserscheide	22
6.4	Anwendung der Wasserscheidentransformation	23
6.5	Vorgehen beim Mean-Shift-Clustering	24
6.6	Anwendung des Mean-Shift-Clustering an einem Bild	24
6.7	Schematische Darstellung von Thresholding	25
6.8	Canny-Algorithmus im Vergleich zu LoG und Sobel-Operator	27
6.9	Schrittweise Kantenerkennung eines Bildes durch Canny Algorithmus	29
6.10	Erkennung der Kante anhand der Intensitätsfunktion	29
6.11	Wechsel der Diskontinuität	30
6.12	Zweite Ableitung zur Erkennung von Kanten	30
6.13	Darstellung des BGP Prozess	32
6.14	Der Effekt des Opening Operators	33
6.15	Der Effekt des Closing Operators	33
6.16	Ablauf der Energievariation	34
6.17	Erstellung der Grauwertematrix	35
6.18	Lineare Verzerrung und Rauschstörung bei der Bilddurch einen Entzerrer	37
6.19	Abbildung des Gauß-Filters im dreidimensionalen Raum	39
6.20	Anwendung des Gauß-Filters mit verschiedenen Varianzen	40
7.1	Grober Programmablauf	43
7.2	Hauptprogrammablauf	45
7.3	Benutzeroberfläche von ImageRepair	46
7.4	Fehlermeldung	46
8.1	Bild mit Graubereich	47
8.2	Bild mit gestreiftem Bereich	48
8.3	Bild mit einfarbigem Bereich	48

Tabellenverzeichnis

2.1	Bildschädenverteilung	3
5.1	Bibliotheken für die Bildverarbeitung im Vergleich.	18
5.2	Vergleich der Sprachen Java, Python und C++	19

1 Einleitung

1.1 Motivation

Nach der erfolgreichen Datenrettung von defekten Datenträgern stellt man häufig fest, dass die Daten irreparabel beschädigt sind. Grund dafür sind häufig Speichersektoren, die durch einen Defekt oder eine schlechte Verbindung nicht mehr auslesbar sind. Besonders bei Bildern ist dies durch eine Veränderung an deren Struktur oder Farbe zu erkennen. Der Beschädigungsgrad ist von Bild zu Bild unterschiedlich. Dies erlaubt, manche Fotos nach der entsprechenden Reparatur zu behalten. Da die Bildermenge auf einem Datenträger beliebig groß sein kann, ist es sinnvoll, diesen Arbeitsschritt zu automatisieren. So bekommen Auftraggeber einer Datenrettung in kürzerer Zeit einen möglichst fehlerfreien Datensatz.

1.2 Ziel der Arbeit

Ziele dieser Arbeit sind folgende:

1. Verbesserung des Kenntnisstandes der Projektteilnehmerinnen durch die Untersuchung der Technologien und Algorithmen für den praktischen Einsatz.
2. Entwicklung einer Anwendung, die den Grad der Beschädigung eines Bildes analysieren, eine Reparatur durchführen und große Menge an Bildern bearbeiten kann.

1.3 Gliederung der Arbeit

Die Arbeit gliedert sich in 9 Kapitel, wobei das Kapitel 1 die Einleitung darstellt. Diese soll einen groben Überblick über das Thema und die Problemstellung geben. In Kapitel 2 sollen die Aufgabenstellung und die Anforderungen an die Applikation näher erläutert werden. Kapitel 3 erläutert Grundlagen der Bildverarbeitung sowie das Auftreten von Fehlern in Bildern. Darauf folgen verwandte Arbeiten, in denen bereits Teile der aktuellen Aufgabenstellung gelöst wurden. Im Anschluss folgen ein Kapitel zur Analyse möglicher Bibliotheken zur Bildverarbeitung sowie ein Kapitel zu möglichen Algorithmen, die für die spätere Implementierung der Anwendung relevant sind. In Kapitel 7 wird der Ablauf der Anwendung und ihre Anforderungen ausführlich beschrieben. Im darauf folgenden Kapitel 8 werden die Ergebnisse der Testdurchläufe evaluiert und die Arbeit wird reflektiert. Zum Schluss folgt eine Zusammenfassung und im Ausblick wird besprochen, wie man die Anwendung noch verbessern könnte.

2 Aufgabenstellung

In dem Kapitel wird über typische Bildfehler, deren Ursachen und die Aufgabenstellung gesprochen, die aus der mit Datenrettung verbundenen Tätigkeit hervorgeht.

2.1 Typische Bildfehler und deren Ursache

Verwackelte Kamera, schlechte Beleuchtung oder ungerade Horizonte – diese und einige andere Gründe verschlechtern die Bildqualität. Im Rahmen dieser Arbeit werden aber Bildfehler anderer Herkunft betrachtet. Es geht um Bilder, die durch einen technischen Schaden Fehler enthalten, die Teile des Bildes unkenntlich machen. Folgende Gründe kommen am häufigsten vor:

- Falscher Einbau oder Ausbau der Speicherkarte.
- Unvollständige Dateiübertragung durch Entfernen der Speicherkarte, während das Gerät noch Lese- / Schreibvorgänge ausführt.
- Fehlerhafte Sektoren auf dem Speichergerät.
- Physische Beschädigung des Speichergeräts.
- Imageschaden durch einen böswilligen Virenangriff.
- Beschädigung des Dateisystems.

Nicht immer ist der Speicherträger Schuld an der Anzeige korrupter Bilder. Auch das Lesegerät, die Verbindung zum Rechner (z.B. Ports) oder auch der Rechner selbst können Probleme verursachen. Das Nutzen anderer Hardware oder auch anderer Software zur Bildanzeige kann solche Anzeigefehler beheben. [1]

Beschädigte Bilder weisen typischerweise folgende Mängel auf:

- Bild unterteilt in falsch angeordnete Abschnitte (siehe Abb. 2.1 [eigene Quelle])
- Einfarbige Bereiche (siehe Abb. 2.2 [eigene Quelle])
- Bereiche mit veränderter Farbe (siehe Abb. 2.3 [eigene Quelle])
- Gestreifte Bereiche (siehe Abb. 2.4 [eigene Quelle])

Laut eigener Statistik hat ein Bild oft eine Mischung aus verschiedenen Schäden. Für die Statistik wurden beschädigte Bilder, die Realfälle erfolgreicher Datenrettungen darstellen, kategorisiert und gezählt. Die Tabelle 2.1 [eigene Quelle] zeigt eine Aufstellung der entdeckten Schäden bei 250 Bildern nach Art des Schadens.



Abbildung 2.1: Bild unterteilt in Segmente



Abbildung 2.2: Bild enthält einen Graubereich



Abbildung 2.3: Bildteil weist andere Farbe auf

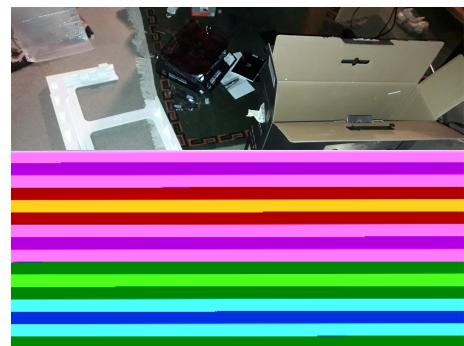


Abbildung 2.4: Bild enthält einen gestreiften Bereich

Art des Schadens	Anzahl des Bilder (St.)
Bild unterteilt in falsch angeordnete Abschnitte	5
Einfarbige Bereiche	65
Einfarbige Bereiche (+veränderte Farbe)	63
Bereiche mit veränderter Farbe	5
Pixel verschoben	1
Bild unterteilt in falsch angeordnete Abschnitte (+ Pixel verschoben) + veränderte Farbe	70
Bild unterteilt in falsch angeordnete Abschnitte (+ Pixel verschoben) + veränderte Farbe (+ Einfarbiger Bereich)	24
Bild unterteilt in falsch angeordnete Abschnitte + Pixel verschoben	12
Bild enthält Teile von anderen Bildern + einfarbigen Bereich	5
Gesamtzahl	250

Tabelle 2.1: Bildschädenverteilung

Offensichtlich kommen die Bilder mit einfarbigen Bereichen und mit falsch angeordneten Abschnitten am häufigsten vor. Daher wird zum Ziel gesetzt, zu versuchen, die auf diese Art geschädigten Bildbereiche zu reparieren oder zu beseitigen.

2.2 Aufgabenstellung

Während der Datenrettung werden von dem beschädigten Datenträger sowohl Bilder als auch andere Dateien ausgelesen. Diese Bilder weisen oft Schäden auf. Es soll zunächst eine umfangreiche Recherche und Analyse durchgeführt werden, um herauszufinden, wie solche Bilder repariert werden können, und welche Möglichkeiten dafür unterschiedliche Programmiersprachen und Bibliotheken bieten. Anhand der Ergebnisse der Recherche soll entschieden werden, welche Technologien angewendet werden können.

Die zu entwickelnde Anwendung ist für die Entfernung der korrupten Bereiche von wiederhergestellten Bildern gedacht. Dafür soll sie beschädigte Bereiche erkennen und diese komplett beseitigen. Falls kein solcher Bereich entdeckt wird, soll das Originalbild separat abgespeichert werden. Da das Zuschneiden der Bilder bislang händisch oder bei einer zu großen Menge an Bildern gar nicht erfolgt, bringt das Automatisieren dieses Arbeitsschrittes eine große Zeitsparnis mit sich. Die Anwendung wird speziell für die Datenrettung mittels professioneller Software von ACELab entwickelt, sodass die Bilder nach dem Abspeichern direkt mit der zu entwickelnden Anwendung zugeschnitten werden können. Da bei der Datenrettung hauptsächlich Bilder im JPEG Format beschädigt sind, wird im folgenden Kapitel das Aufkommen von Fehlern auf JPEG Bilder bezogen.

3 Bildverarbeitung

In diesem Teil der Arbeit werden Grundlagen zu digitalen Bildern sowie das Image Processing behandelt.

3.1 Bilder im digitalen Kontext

Ein Bild ist eine zweidimensionale Funktion $f(x,y)$, die einem Koordinatenpaar einen Wert zuweist. Dieser Wert stellt die Farbe bzw. Intensität des Pixels an den Koordinaten x und y dar. Farbige Bilder werden durch die drei Kanäle Rot, Grün und Blau (RGB) dargestellt. Der Pixelwert farbiger Bilder wird daher durch ein 3-Tupel (Tripel) dargestellt, das die RGB- Werte repräsentiert. Bei Graustufenbildern genügt ein Kanal, so kann auch der Pixelwert mit nur einer Zahl dargestellt werden. Um ein Bild auf einem Computer bearbeiten zu können, muss es digital vorliegen. Die Daten zum Darstellen des Bildes können in ein multidimensionales Array extrahiert werden, um es später bearbeiten zu können. Für farbige Bilder werden dreidimensionale Arrays, bei Graustufenbilder nur zwei Dimensionen benötigt [2]. Abbildung 3.1 [3] stellt das dreidimensionale Array eines farbigen Bildes dar. Das Array setzt sich zusammen aus Breite und Höhe des Bildes, multipliziert mit den drei Farbkanälen.

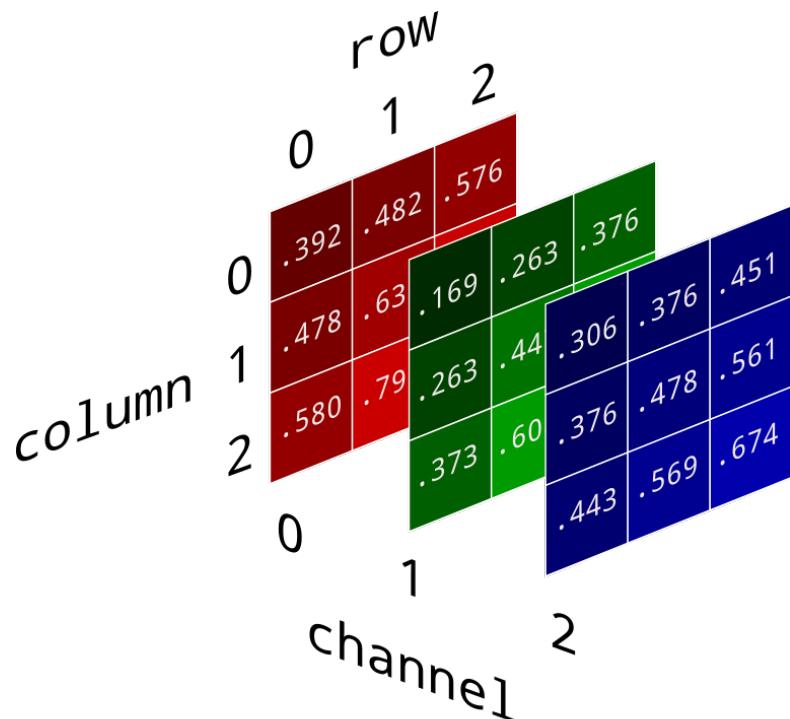


Abbildung 3.1: Aufbau eines Farbbildes in seinen Pixelarrays

3.2 Farträume und Farbmodelle

Farbmodelle dienen der korrekten Darstellung von Farbinformationen. Sie basieren auf einem Farbraum oder stellen selbst einen gesamten Farbraum dar. Farträume werden aus 3 Basismengen gebildet, daher ist ein Farbraum immer ein dreidimensionales Koordinatensystem. Die drei Koordinatenachsen stellen die 3 Basismengen dar, jede Farbe wird durch einen Punkt im Koordinatensystem repräsentiert. Ein Farbmodell nutzt drei Primärfarben, um dadurch jede beliebige Farbe im zugehörigen Farbraum abbilden zu können. Farbwerte aus einem Farbraum können durch Anwendung einer linearen Transformation in einen anderen Farbraum konvertiert werden.

Neben RGB, einem der bekanntesten Farträume, gibt es weitere Modelle wie z.B. Standard-RGB (sRGB), Cyan-Magenta-Yellow (CMY), Cyan-Magenta-Yellow-Key (CMYK), Hue Saturation Intensity (HSI) oder HSV. Im Folgenden werden die Farträume RGB und HSV näher beschrieben. [4]

3.2.1 RGB

Der RGB-Farbraum ist struktuell ähnlich zum menschlichen Sehsystem. Gemäß der Struktur des menschlichen Auges, können alle Farben durch eine Kombination der Basisfarben Rot, Grün und Blau dargestellt werden. In der Abbildung 3.2 [5] wird der Farbraum in einer Würfelform dargestellt. Die Farben Rot, Grün und Blau bilden die Koordinatenachsen. Der Ursprung bildet die Farbe Schwarz ab, Weiß wird am davon weitesten entfernten Eck geformt. Die Gerade von Schwarz zu Weiß bildet die Grauachse. Jeder Punkt kann durch seinen Ortsvektor dargestellt werden. Modelle des RGB-Farbraums werden bei der Bilderstellung z.B. durch Kameras und bei der Bildanzeige auf Displays eingesetzt. [4]

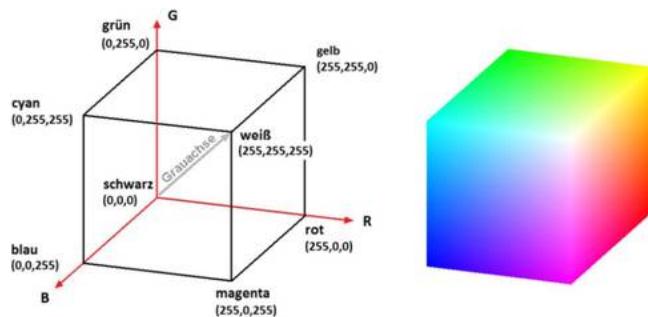


Abbildung 3.2: RGB Farbraum

3.2.2 Hue Saturation Value(HSV)

HSV-Modelle erfüllen die Eigenschaften der optischen Wahrnehmung des Menschen und sind geeignet für farbbasierte Bildverarbeitung. Darüber hinaus sind diese unabhängig von Geräten zur Bildanzeige. Im Gegensatz zum RGB-Farbraum werden hier keine Farben als Basiswerte eingesetzt, sondern der Farbton (Hue), die Sättigung (Saturation) und die Helligkeit (Density/Brightness Value).

Das menschliche Sehsystem nutzt diese drei Charakteristiken zur Beschreibung von Farben. Der Farbton ist ein wichtiger Faktor in der Farbzusammensetzung, besonders in der Kunst. Er wird mit der dominanten Wellenlänge der spektralen Energieverteilung in Verbindung gebracht.

Abbildung 3.3 [6] zeigt den Farbraum in Zylinderform. Die Mittelachse stellt die Grauachse dar, die die beiden Kreisflächen miteinander verbindet. Die untere Kreisfläche bildet aufgrund der fehlenden Helligkeit die Farbe Schwarz ab, die obere Kreisfläche hingegen aufgrund der vollen Helligkeit Weiß. Abbildung 3.4 [7] zeigt die Querschnittsfläche auf den Zylinder. In dieser Ansicht wird die Helligkeit ausgenommen. Der Winkel ausgehend von der Grauachse zeigt den Farbton an. Die Sättigung wird durch die Länge des Vektors bestimmt.

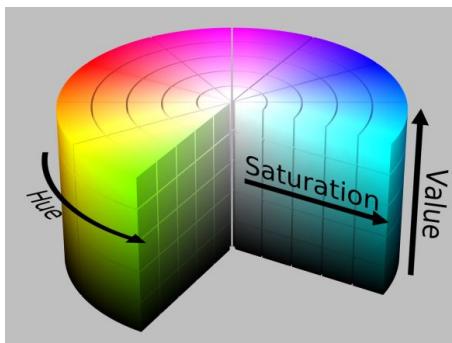


Abbildung 3.3: HSV Farbraum

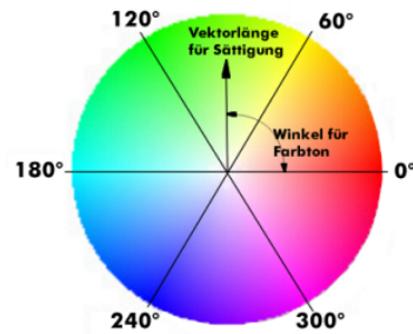


Abbildung 3.4: Winkel zur Bestimmung des Farbtos

Für den HSV-Farbraum gibt es weitere Modelle, wie z.B. der nach unten gerichtete Kegel, die Halbkugel oder die hexagonale Pyramide. [4]

3.3 Fehler in JPEG Bildern

Bilder im JPEG Format bestehen aus Blöcken, genannt Minimum Coded Unit (MCU), die standardmäßig aus 64 oder 128 Pixeln bestehen. Ein Fehler in einem oder mehreren dieser Blöcke führt dazu, dass die folgenden MCU-Blöcke korrupt werden. Dadurch entstehen Farbtonwechsel (vlg. Abbildung 2.3), Veränderungen in der Helligkeit im restlichen Bild oder Verschiebungen der Blöcke bis zum Ende des Bildes. [8]

Auch größere Bereiche in Grau oder einer anderen einheitlichen Farbe werden durch korrupte Blöcke verursacht. Bits ergeben dabei unabsichtlich einen JPEG Marker, wodurch das Bild nur bis zu diesem Punkt angezeigt wird (vlg. Abbildung 2.2). Abbildung 3.5 [9] stellt den Aufbau und die verschiedenen Marker dar. Ein JPEG Bild beginnt mit dem Start of Image (SoI) Marker, worauf die verschiedenen Metadaten folgen, z.B. Datum der Aufnahme, Kameramodell und -einstellungen. Der Start of Scan (SoS) Marker markiert den Anfang des Bildes, End of Image (EoI) markiert das Ende des Bildes. Eine weitere Ursache für fehlerhafte JPEG Bilder sind fehlende Marker im Bereich des Headers. Fehlen Marker, können Anwendungen das betroffene Bild nicht mehr anzeigen. [9]

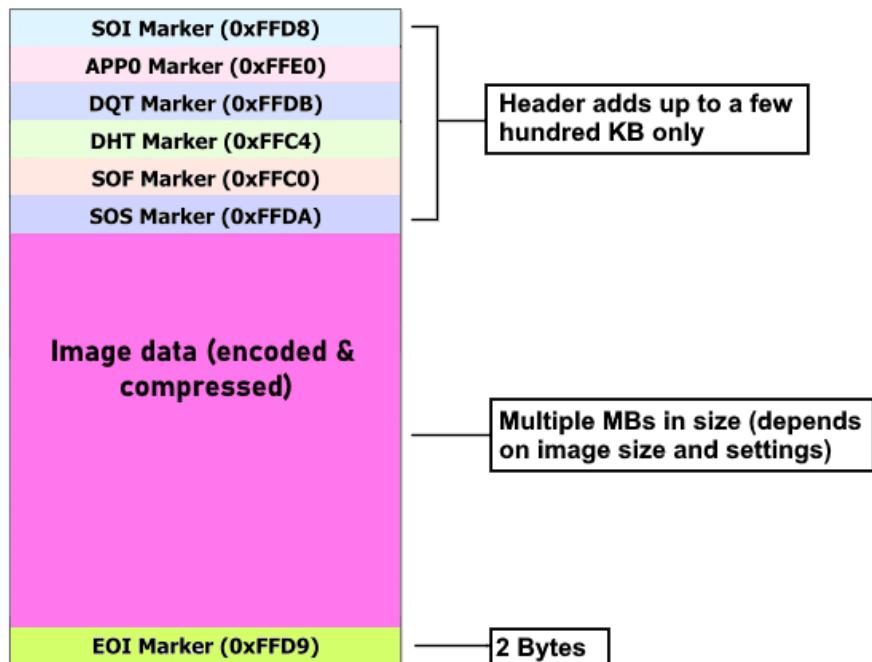


Abbildung 3.5: Aufbau des JPEG Formats

3.4 Ansätze für die Bildreparatur

Beschädigte MCU-Blöcke können nach Identifizierung gelöscht werden, um die Verschiebung aufzuheben. Die Veränderung in den Farb- oder Helligkeitswerten können manuell durch Ausprobieren an den Originalteil des Bildes angepasst werden, sodass das Bild wieder dem Originalbild ähnelt. Fehlende Marker im Headerbereich können durch einen intakten Headerbereich eines ähnlichen Bildes ersetzt werden, das mit derselben Kamera und Auflösung geschossen wurde. Die Bilddaten nach dem SoS Marker bleiben erhalten und das Bild kann wieder angezeigt werden. [9]

3.5 Ablauf der Bildverarbeitung

Image Processing ist das automatische Verarbeiten, Analysieren, Bearbeiten und Interpretieren von Bildern auf dem Computer mithilfe von Algorithmen und Code. Abbildung 3.6 [2] zeigt den Ablauf der grundlegenden Schritte, die nacheinander durchgeführt werden [2]

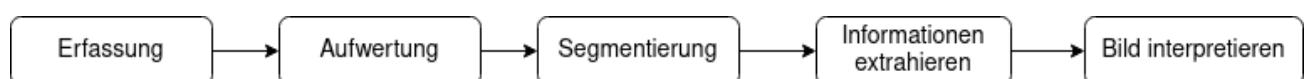


Abbildung 3.6: Ablauf der Bildverarbeitungsschritte

Erfassung

Bei der Erfassung wird das Bild beispielsweise mit einer Kamera digitalisiert und auf einem digitalen Speicher, z.B. einer Secure Digital Memory Karte (SD-Karte) als Bilddatei abgespeichert.

Aufwertung

Die Bildaufwertung beinhaltet die Manipulation des Bildes. Das Bild wird z.B. durch passende Filteralgorithmen aufgewertet und restauriert.

Segmentierung

Bei der Segmentierung wird das Bild zugeschnitten, um den gewünschten Teil des Bildes zu extrahieren und später analysieren zu können.

Informationen extrahieren

In diesem Schritt werden technische Informationen zum extrahierten Teil des Bildes gewonnen und das Bild in passender Form repräsentiert.

Bild interpretieren

Bei der Interpretation des Bildes werden beispielsweise mithilfe der Objekterkennung Objekte erkannt um daraus Wissen zu generieren. [2]

3.6 Zusammenfassung

In diesem Abschnitt werden die Grundlagen zu digitalen Bildern und der Bildverarbeitung kurz zusammengefasst. Bilder werden mithilfe eines oder mehrerer Arrays digital gespeichert, je nachdem ob es sich um ein Graustufenbild oder ein farbiges Bild handelt. Zur Darstellung von Farben werden Farbräume genutzt, die auf 3 Farben basieren. Bekannte Farbräume sind der RGB-Farbraum sowie der HSV-Farbraum, der häufig bei der Bildverarbeitung eingesetzt wird. Im Anschluss wurden Fehler in Bildern im JPEG Format untersucht, da dieses Format am verbreitetsten ist. Fehler in der Anzeige können durch fehlende Marker im Headerbereich auftreten oder auch durch das Verschieben von MCU Blöcken. Ein Bild kann durch einen Bitfehler möglicherweise nicht komplett angezeigt werden, wenn dieser als EoF Marker interpretiert wird. Diese Fehler können beispielsweise durch manuelle Berichtigung der MCU Blöcke oder durch Ersetzen des Headers mit einem intakten Header verhindert werden. Zum Schluss wurde der Ablauf der Bildverarbeitung betrachtet: Nach der Erfassung und Digitalisierung des Bildes wird dieses zuerst durch Filter aufgewertet und der gewünschte Bereich wird segmentiert. Danach können Informationen extrahiert werden, mit deren Hilfe eine Interpretation des Bildes stattfinden kann. Für die Arbeit werden die Schritte Aufwertung, Segmentierung und Extrahierung von Informationen relevant sein.

4 Verwandte Arbeiten

Bildverarbeitung ist ein großer Forschungsbereich. Mit der Verbreitung der digitalen Bilder hat ihre Bedeutung zugenommen. In den folgenden Abschnitten werden die für das untersuchte Thema relevanten Bücher und wissenschaftlichen Arbeiten vorgestellt.

4.1 Fachliteratur

Das Werk von Yu-Jin Zhang "Handbook of Image Engineering" [4] definiert allgemeine Konzepte, verwandte Prinzipien, praktische Techniken und spezifische Methoden der Bildverarbeitung. In dem Buch sind Erklärungen und Einsatzbereiche unterschiedlichster Algorithmen zur Merkmalerkennung zusammengefasst, wodurch man sie schneller finden und besser verstehen kann.

Arcangelo und Cosimo Distante [10] haben ein fundamentales Handbuch "Handbook of Image Processing and Computer Vision" in drei Teilen erstellt, in denen folgende Aspekte behandelt werden:

- grundlegende physikalische Prozesse der Umwandlung eines analogen Signals ins digitale
- wichtigste Charakteristika der digitalen Bilder und ihre Verwendung in der Bildverarbeitung
- Wiederherstellung der geschädigten Bilder
- Objekterkennung und -analyse mit neuronalen Netzen
- 3D Rekonstruktion und anderes.

Aus dem Buch wurden Erklärungen zu Canny Algorithmus, Filtern und Laplacian Operator entnommen, welche zu deren besserem Verständnis beigetragen haben und den Einsatz von diesen Methoden bei der Entwicklung der ImageRepair Anwendung ermöglicht haben.

4.2 Wissenschaftliche Arbeiten

In zahlreichen wissenschaftlichen Arbeiten werden unterschiedliche Probleme der Bildverarbeitung angesprochen. Für die aktuelle Arbeit sind jedoch die relevant, die den Einsatz von Algorithmen zeigen. Nobuyuki Otsu hat in "A threshold selection method from gray level histograms" [11] sein berühmtes Verfahren für die Bestimmung des Schwellenwerts vorgestellt. Textur- und Kantenerkennung sind wichtige Methoden in der Bildverarbeitung und sind entsprechend in "Binary Gabor pattern: An efficient and robust descriptor for texture classification" [12] und "Texture Classification Approach Based on Energy Variation" [13] besprochen. Filtering Methoden kommen auch sehr oft zum Einsatz. Die Besonderheiten von dem Eigenfilter werden in "On The Eigenfilter Design Method and Its Applications: A Tutorial" [14] besprochen, und die Autoren von "Constrained least squares filtering

followed by denoising of decomposed images using wave atom and wavelet transform" [15] beleuchten den Mechanismus des Filters, der die Methode der kleinsten Quadrate verwendet.

Für die Funktionalität der Anwendung im Rahmen dieser Arbeit spielen die Algorithmen für die Kantenerkennung und das Verfahren für die Bestimmung des Schwellenwerts eine wichtige Rolle, denn sie dienen zur Erkennung der beschädigten Bereiche eines Bildes.

5 Analyse der Bibliotheken

Bibliotheken für Bildverarbeitung können in Web- oder Desktop-Applikationen verwendet werden. Viele von ihnen bieten ähnliche Funktionen an. In diesem Kapitel werden die Bibliotheken von den drei gängigsten Programmiersprachen für Bildverarbeitung betrachtet: JavaScript, C++ und Python. Für die Reparatur geschädigter Bilder ist es zunächst wichtig, einen fehlerhaften Bereich zu erkennen. Danach soll das Bild zugeschnitten und gespeichert werden. Falls kein solcher Bereich entdeckt wird, soll das Orginalbild separat abgespeichert werden. Die Bibliotheken werden im Bezug auf die Existenz der notwendigen Funktionen betrachtet.

5.1 JavaScript

In diesem Abschnitt werden die beiden JavaScript Bibliotheken sharp.js und CamanJS und die Funktionen zur Bildverarbeitung näher untersucht.

5.1.1 sharp.js

Node.js verfügt über ein Ökosystem von Bibliotheken, die für die Verarbeitung von Bildern verwendet werden können, wie, zum Beispiel, das sharp-, jimp- und gm-Modul. sharp ist eine der beliebtesten Node.js-Bildverarbeitungsbibliotheken. [16]

Der typische Anwendungsfall für sharp ist die Konvertierung großer Bilder in gängigen Formaten in kleinere, webfreundliche JPEG-, Portable Network Graphics (PNG)-, AV1 Image File Format (AVIF)- und WebP-Bilder unterschiedlicher Größe. Farbräume, eingebettete International Color Consortium (ICC)-Profile und Alpha-Transparenzkanäle werden alle korrekt verarbeitet. Lanczos Resampling stellt sicher, dass die Qualität nicht der Geschwindigkeit geopfert wird. Neben der Größenänderung von Bildern sind auch Operationen wie Rotation, Extraktion, Compositing und Gammakorrektur möglich. Die meisten modernen macOS-, Windows- und Linux-Systeme, auf denen Node.js $\geq 12.13.0$ läuft, benötigen keine zusätzlichen Installations- oder Laufzeit-Abhängigkeiten. Dieses Modul unterstützt das Lesen von JPEG, PNG, WebP, AVIF, Tagged Image File Format (TIFF), Graphics Interchange Format (GIF) und Scalable Vector Graphics (SVG)-Bildern. Die Ausgabe von Bildern kann in den Formaten JPEG, PNG, WebP, AVIF und TIFF sowie als unkomprimierte Pixel-Rohdaten erfolgen. Streams, Pufferobjekte und das Dateisystem können für die Ein- und Ausgabe verwendet werden. Ein einzelner Input-Stream kann in mehrere Verarbeitungspipelines und Output-Streams aufgeteilt werden. [17]

sharp hat libvips als Bildverarbeitungsengine. Das ist eine Bibliothek, die über etwa 300 Operationen verfügt, darunter Arithmetik, Histogramme, Faltung, morphologische Operationen, Frequenzfilterung, Farbe, Resampling, Statistik und andere. libvips unterstützt eine große Anzahl von numerischen

Typen, von 8-bit int bis 128-bit complex. Bilder können eine beliebige Anzahl von Bändern haben. Es unterstützt eine Vielzahl von Bildformaten, darunter JPEG, JPEG2000, JPEG-XL, TIFF, PNG, WebP, High Efficiency Image File Format (HEIC), AVIF, Flexible Image Transport System (FITS), Matlab, OpenEXR, Portable Document Format (PDF), SVG, High Dynamic Range (HDR), Portable Pixmap (PPM) / Portable Graymap (PGM) / Portable FloatMap (PFM), Comma-Separated Values (CSV), GIF, Analyze, Neuroimaging Informatics Technology Initiative (NIfTI), DeepZoom und OpenSlide. Es kann auch Bilder über ImageMagick oder GraphicsMagick laden, so dass es mit Formaten wie Digital Imaging and Communications in Medicine (DICOM) arbeiten kann. [18]

Nützliche Funktionen von sharp hinsichtlich der Bilderreparatur sind die Größenänderung von Bildern und Farbmanipulation. Die Letztere erfolgt mithilfe von dem npm-Paket `color`, das erlaubt, Farben durch ihre Hashwerte zu erkennen. [19]

5.1.2 CamanJS

CamanJS ist (ca)nvas (man)ipulation in Javascript. Es ist eine Kombination aus einer einfach zu bedienenden Benutzeroberfläche mit fortschrittlichen und effizienten Bild/Canvas-Bearbeitungstechniken. CamanJS kann mit neuen Filtern und Plugins erweitert werden, und es kommt mit einer breiten Palette von Bildbearbeitungsfunktionen, die weiter wachsen. Es ist völlig bibliotheksunabhängig und funktioniert sowohl in NodeJS als auch im Browser. [20]

Die Verwendung von CamanJS ist extrem einfach, aber auch sehr flexibel. Der Zugriff auf die gesamte Funktionalität erfolgt durch den Aufruf der Funktion `Caman()`. [21]

CamanJS verfügt über grundlegende integrierte Filter. So können Helligkeit, Kontrast, Sepia, Sättigung, Belichtung, Rauschen, Schärfe, Vibrance und Farbton direkt angewendet werden. Einige Filter wie Helligkeit und Kontrast können sowohl einen negativen als auch einen positiven Wert haben. Alle diese Filter sind in CamanJS in der Kernbibliothek enthalten. [22]

Mit CamanJS können Bilder zugeschnitten und vergrößert/verkleinert werden. Dies kann entweder bei der Initialisierung (durch direkte Größenangabe) oder zu einem späteren Zeitpunkt geschehen. Zu beachten ist, dass nur die Größenänderung bei der Initialisierung in der Kernbibliothek enthalten ist. Es muss entweder die Vollversion von CamanJS verwendet werden oder das Größen-Plugin manuell eingebunden werden, wenn man später zuschneiden oder die Größe ändern möchte. Es gibt mehrere Möglichkeiten, die Kernfunktionalität von CamanJS zu erweitern. Der größte Teil von CamanJS besteht aus Filtern, die pixelweise arbeiten. Der Filterfunktion werden die RGB-Werte eines einzelnen Pixels übergeben, und die Funktion gibt die aktualisierten Werte zurück. Diese Filter haben auch den Zugriff auf ein spezielles Objekt, das die aktuelle Position des Pixels kennt, das der Verarbeitungsfunktion übergeben wurde. Dieses Objekt stellt viele hilfreiche Funktionen zur Verfügung, mit denen nicht nur die Position, sondern auch die Werte anderer Pixel im Bild abgerufen beziehungsweise eingestellt werden können. Nicht alles lässt sich pixelweise erledigen. Eine andere Möglichkeit, Bilder zu verändern, ist die Verwendung eines Convolution Kerns. Dabei handelt es sich um eine Matrix,

die beschreibt, wie ein bestimmtes Pixel auf der Grundlage der umliegenden Pixel verändert werden soll. In CamanJS werden die Kernel durch 1-dimensionale Arrays definiert. Wenn das, was gemacht werden soll, nicht in den Arbeitsablauf von pixelweisen Plugins oder Kernelmanipulationen passt, kann man auch mit Hilfe von Plugins einen vollen Zugriff auf die Bilddaten innerhalb von CamanJS erhalten. Einige Beispiele für Plugins sind der Stack Blur Algorithmus und Cropping/Resizing. [21]

5.2 Python

Python ist die Programmiersprache, die die größte Anzahl der Bibliotheken für Bildverarbeitung hat. Das kann dadurch erklärt werden, dass Python für die Aufbereitung und Verarbeitung großer Datensets und für Machine Learning Algorithmen sehr gut geeignet ist. Die beliebtesten Bibliotheken sind OpenCV und Scikit-Image. Diese werden im Folgenden vorgestellt.

scikit-image (`skimage`) ist eine Open-Source Bibliothek für Python, die für viele Aufgaben der Computer Vision nützlich sein kann, enthält aber hauptsächlich fortgeschrittene Funktionen und bildspezifische Routinen. Grundlegende Bildmanipulationen, wie zum Beispiel das Zuschneiden von Bildern oder einfaches Filtern, kann mit NumPy und SciPy realisiert werden. Das Entwicklerteam empfiehlt scikit-image zusammen mit den Letzteren zu benutzen. [23]

Vor der Installation von scikit-image sollen Numpy und SciPy installiert werden. Das Importieren von Bildern ist der allererste Schritt. Das Bild und seine Intensität werden in Form von Zahlen gespeichert. Die Zahlen werden als Pixel definiert. [24]

Die Bibliothek arbeitet mit jedem Bildformat, das von Python Imaging Library unterstützt wird oder mithilfe eines Plugins sichergestellt wird. Bildarrays können entweder durch Ganzzahlen (mit oder ohne Vorzeichen) oder durch Fließkommazahlen dargestellt werden. [23]

In `skimage` sind Bilder Numpy-Arrays, die eine Vielzahl von Datentypen, das heißt "dtypes", unterstützen. Um eine Verzerrung der Bildintensitäten zu vermeiden (bei Skalierung von Intensitätswerten), wird davon ausgegangen, dass Bilder die in der Abbildung 5.1 [25] aufgelisteten dtype-Bereiche verwenden.

Float-Bilder sollten auf den Bereich -1 bis 1 beschränkt werden, auch wenn der Datentyp selbst diesen Bereich überschreiten kann; alle Integer dtypes hingegen haben Pixelintensitäten, die den gesamten Datentypbereich umfassen können. Bis auf wenige Ausnahmen werden 64-Bit-(u)int-Bilder nicht unterstützt. Die Funktionen in `skimage` sind so konzipiert, dass sie jeden dieser dtypes akzeptieren, aber aus Effizienzgründen ein Bild eines anderen dtype zurückgeben können. Wenn ein bestimmter dtype benötigt wird, stellt `skimage` Dienstprogramme zur Verfügung, die dtypes konvertieren und die Bildintensitäten korrekt neu skalieren. [25]

Da Bilder in scikit-image durch NumPy ndarrays (N-dimensionales Array, hat in der Regel eine feste Größe) dargestellt werden, können viele gängige Operationen mit Standard-NumPy-Methoden zur

Data type	Range
uint8	0 to 255
uint16	0 to 65535
uint32	0 to $2^{32} - 1$
float	-1 to 1 or 0 to 1
int8	-128 to 127
int16	-32768 to 32767
int32	-2^{31} to $2^{31} - 1$

Abbildung 5.1: dtype-Bereiche

Bearbeitung von Arrays durchgeführt werden, beispielsweise Abrufen der Geometrie des Bildes und der Anzahl der Pixel oder Abrufen von statistischen Informationen über die Bildintensitätswerte. Das NumPy-Indexing kann sowohl für die Anzeige der Pixelwerte als auch für deren Änderung verwendet werden. Über einzelne Pixel hinaus ist es möglich, auf die Werte ganzer Pixelmengen zuzugreifen und diese zu ändern, indem man die verschiedenen Indizierungsmöglichkeiten von NumPy nutzt. Masken sind sehr nützlich, wenn eine Gruppe von Pixeln ausgewählt werden muss, an denen Manipulationen vorgenommen werden sollen. Die Maske kann ein beliebiges boolesches Array sein, das die gleiche Form wie das Bild hat (oder eine Form, die auf die Bildform übertragen werden kann). Damit lässt sich ein interessierender Bereich definieren. Das oben Gesagte gilt auch für Farbbilder. Ein Farbbild ist ein NumPy-Array mit einer zusätzlichen Dimension für die Kanäle. Einzelne Pixelwerte können gelesen und gesetzt werden. [26]

Da Bilder NumPy-Arrays sind, kann das Zuschneiden eines Bildes mit einfachen Slicing-Operationen durchgeführt werden. Slicing ist das Aufteilen des Gesamtbildes in kleinere Bilder. [27]

Eine hilfreiche Funktion von scikit-image hinsichtlich der Reparatur geschädigter Bilder wäre die Bildsegmentierung. Es geht darum, die Pixel von interessierenden Objekten in einem Bild zu kennzeichnen. Somit könnte man vermutlich die beschädigten Bereiche eines Bilds für die weitere Bearbeitung markieren und gruppieren. Mit der Segmentierung können die Konturen eines Bereichs abgegrenzt werden. Die Methode hat aber einen Nachteil: wenn die Konturen des Objektes nicht geschlossen werden können, kann es nicht komplett abgegrenzt und dadurch nicht ausgeschnitten werden. Eine weitere Methode ist die Segmentierung nach Regionen, für die die Wasserscheidentransformation

(Watershed) benutzt wird. Näher wird der Algorithmus im Kapitel 6 beschrieben. Diese Methode lässt die Grenzen von Objekten genauer erkennen als die normale Segmentierung. [28]

Der Random-Walker-Algorithmus ähnelt dem Watershed, verfolgt aber einen "probabilistischeren" Ansatz. Er basiert auf der Idee der Diffusion von Labels im Bild. Die `measure`-Funktion von scikit-image lässt die Größe und den Umfang der segmentierten Bereiche messen. Dies kann hilfreich sein, wenn eine Entscheidung getroffen werden muss, ob ein beschädigtes Bild reparabel ist. [23]

scikit-image verfügt über Methoden zur Merkmalerkennung und -extraktion. Darunter sind Eckenerkennung, Bestimmen der Grauwertmatrix, Objekterkennung, Gesichtserkennung und andere. [29]

5.3 C++

C++ ist eine Mehrparadigmensprache, die es ermöglicht, auf einfache Weise effiziente Bildverarbeitungsalgorithmen zu erstellen. Das ist eine High-Level-Sprache, die erlaubt, leistungsstarke Abstraktionen zu entwickeln und verschiedene Programmierstile zu mischen, um die Entwicklung zu erleichtern. Gleichzeitig ist C++ Low-Level und kann die Vorteile der Hardware voll ausschöpfen, um die beste Leistung zu liefern. Außerdem ist es sehr portabel und hochkompatibel, so dass Algorithmen aus High-Level-Sprachen wie Python aufgerufen werden können, die ein schnelles Prototyping ermöglichen. [30] In diesem Kapitel wird von Besonderheiten und Funktionalität einer der C++ Bibliotheken - OpenCV - berichtet.

OpenCV ist eine Open-Source-Bibliothek für Computer Vision. Die Bibliothek ist in C und C++ geschrieben und läuft unter Linux, Windows und Mac OS X. Es gibt Schnittstellen für Python und Java. Die OpenCV-Bibliothek enthält über 500 Funktionen, die viele Bereiche der Bildverarbeitung abdecken, darunter Fabrik Produktinspektion, medizinische Bildgebung, Sicherheit, Benutzeroberfläche, Kamerakalibrierung, Stereovision und Robotik. Weil Computer Vision und maschinelles Lernen oft Hand in Hand gehen, enthält OpenCV auch eine vollständige, universell einsetzbare Bibliothek für maschinelles Lernen (Machine Learning Library (MLL)). Diese Unterbibliothek ist auf statistische Mustererkennung und Clustering ausgerichtet. Die MLL ist sehr nützlich für die Bildverarbeitungsaufgaben, die im Mittelpunkt von OpenCVs Mission stehen, aber sie ist allgemein genug, um für jedes maschinelle Lernproblem verwendet zu werden.

OpenCV verwendet andere Datenstrukturen als scikit-image. `ip1Image` ist die Grundstruktur für die Kodierung dessen, was allgemein als "Bild" bezeichnet wird. Diese Bilder können Graustufen-, Farb- oder Vierkanalbilder (RGB+Alpha) sein, und jeder Kanal kann verschiedene Arten von Integer- oder Fließkommazahlen enthalten. Dieser Typ ist also allgemeiner als das allgegenwärtige dreikanalige 8-Bit RGB-Bild. OpenCV bietet ein Arsenal an nützlichen Operatoren, die auf die Bilder wirken, darunter Werkzeuge, um die Größe von Bildern zu ändern, einzelne Kanäle zu extrahieren, den größten oder kleinsten Wert eines bestimmten Kanals zu finden, zwei Bilder zu addieren, einen Schwellenwert

für ein Bild festzulegen und so weiter. Ein weiterer Datentyp heißt `CvMat`, der eine Matrix-Struktur hat. Obwohl OpenCV vollständig in C implementiert ist, ist die Beziehung zwischen `CvMat` und `IpImage` ähnlich wie die Vererbung in C++. In jeder Hinsicht kann ein `IpImage` als von `CvMat` abgeleitet betrachtet werden. Eine dritte Klasse, `CvArr`, kann als eine abstrakte Basisklasse betrachtet werden, von der `CvMat` selbst abgeleitet ist. `CvArr` wird oft in Funktionsprototypen benutzt. OpenCV bietet einen Mechanismus zur Serialisierung und De-Serialisierung seiner verschiedenen Datentypen auf und von der Festplatte entweder im Yet Another Markup Language (YAML)- oder Extensible Markup Language (XML)-Format. Die OpenCV-Funktionen, die die Interaktion mit dem Betriebssystem, dem Dateisystem und Hardware wie z.B. Kameras ermöglichen, sind in einer Bibliothek namens HighGUI (was für "High-Level Graphical User Interface" steht) zusammengefasst. Die HighGUI-Bibliothek kann in drei Teile unterteilt werden: den Hardware-Teil, den Dateisystem-Teil und den GUI-Teil. Der Hardware-Teil befasst sich in erster Linie mit dem Betrieb von Kameras. Der Dateisystemteil kümmert sich um das Laden und Speichern von Bildern. Der dritte Teil von HighGUI ist das Fenstersystem (oder GUI). Die Bibliothek bietet einige einfache Funktionen, die es ermöglichen, ein Fenster zu öffnen und ein Bild in dieses Fenster zu legen. [31]

Genauso wie in scikit-image ist in OpenCV die ganze Reihe von Algorithmen zur Merkmalerkennung implementiert:

- Eckenerkennung (Harris und Shi-Tomasi Algorithmen)
- Scale-invariant feature transform (SIFT).
- Speeded-Up Robust Features (SURF). Das ist eine schnellere Version von SIFT
- Binary Robust Independent Elementary Features (BRIEF). Merkmalerkennung mit weniger Speicherplatznutzung, höherer Erkennungsrate und schnellerem Abgleich
- Für Merkmalsabgleich werden in OpenCV Brute-Force Matcher und Fast Library for Approximate Nearest Neighbors (FLANN) verwendet. Mithilfe von Homography können Objekte in komplexen Bildern erkannt werden. [32]

Neben den genannten Funktionen verfügt OpenCV über die Module für Videoverarbeitung, Kamera Kalibrierung, Rauschunterdrückung, Objekterkennung und maschinelles Lernen. Diese sind für das zu lösende Problem irrelevant, daher werden sie nicht betrachtet.

5.4 Wahl der Sprache und Bibliothek

Im Rahmen des Kapitels wurden die JavaScript-Bibliotheken sharp.js und CamanJS, scikit-image (Python) und OpenCV(C++) untersucht, da diese zu den beliebtesten Bibliotheken für die Bildverarbeitung gehören (laut Online-Ressourcen openbase.com [33], stackoverflow.com [34]). Es wurde festgestellt, dass diese mit verschiedenen Bildformaten arbeiten können. Verwendete Datentypen sind

unterschiedlich, aber die für die Aufgabenstellung relevanten Funktionen wie Merkmalerkennung, pixelweise Analyse und Manipulation, die Möglichkeit ein Bild zuzuschneiden sind in jeder Bibliothek vorhanden. Eigenschaften der untersuchten Bibliotheken, die für das zu erstellende Programm relevant sein könnten, sind in der Tabelle 5.1 [eigene Quelle] dargestellt. Diese Eigenschaften ergeben sich entweder aus den im Kapitel 5 besprochenen Funktionen oder werden in den Dokumentationen zu jeder der genannten Bibliotheken erwähnt.

Funktion	sharp.js (JavaScript)	CamanJS (JavaScript)	scikit-image (Python)	OpenCV (C++/Python)
Bild/Bilder einlesen	x	x	x	x
Bild anzeigen	x	x	x	x
Unterstützung unterschiedlicher Bildformate	x	x	x	x
Konvertierung des Bildformats	x	x	x	x
Konvertierung der Bildfarbe	x	x	x	x
Morphologische Operationen	x	x	x	x
Pixelweise Verarbeitung	x	x	x	x
Merkmalerkennung:				
- Kantenerkennung	-	-	x	x
- Eckenerkennung	-	-	x	x
Merkmalsabgleich	-	-	x	x
Objekterkennung	-	-	x	x
Texturenanalyse	-	-	x	x
Zuschneiden	x	-	x	x
Reparaturfunktionen (siehe Filterungsmethoden im Kapitel 6)	x	-	x	x
Parallelisierung des Verarbeitungsprozesses	x	-	x	x
Methoden des maschinellen Lernens	-	-	-	x

Tabelle 5.1: Bibliotheken für die Bildverarbeitung im Vergleich.

JavaScript kommt oft in der Web-Programmierung zum Einsatz, dadurch sind die typischen Funktionen der beiden betrachteten Bibliotheken bedingt. Übliche Anwendungsfälle sind Konvertierung großer Bilder in kleinere webfreundliche Bilder, Vergrößerung/Verkleinerung/Zuschneiden der Bilder, Verwendung der Filter, Einstellung unterschiedlicher Parameter wie Helligkeit, Kontrast, Sättigung und anderer. Die Funktionalitäten von sharp.js und CamanJS sind jedoch gegenüber zwei anderen untersuchten Bibliotheken beschränkt, obwohl sie ein großes Spektrum der grundlegenden Methoden bieten. Die Erweiterbarkeit von den beiden JavaScript-Bibliotheken durch Plugins macht sie flexibler als bei der Benutzung nur der Kernfunktionalität.

scikit-image basiert auf den wissenschaftlichen Python-Bibliotheken NumPy und SciPy und kann als

eine Erweiterung deren Funktionalität für Bildverarbeitung dienen. Segmentierung, Farbmanipulationen, Filterung, Analyse, Feature Erkennung sind die gängigsten Funktionen von scikit-image.

Wie in der Tabelle 5.1 gezeigt ist, hat OpenCV den größten Umfang von Algorithmen für Bild- und Videoverarbeitung unter den untersuchten Bibliotheken. Die vorhandenen Funktionen von OpenCV erfüllen vollumfänglich die zu Beginn dieses Kapitels aufgeführten Kriterien der Auswahl einer Bibliothek, und zwar die Möglichkeit einen fehlerhaften Bereich im Bild zu erkennen, seine Größe zu analysieren und das Bild zu zuschneiden, falls die Reparatur möglich ist. Zu den besonderen Eigenschaften von OpenCV gehören die Möglichkeit Daten durch die Parallelisierung schnell zu verarbeiten, das Modul für das maschinelle Lernen und Deep Learning. Die Letzteren bringen mit sich die vorteilhaften Funktionen zur Mustererkennung und Bildreparatur, was für die Bearbeitung großer Bildermengen nützlich sein kann. Das sind Faktoren, die OpenCV zu der am besten geeigneten Bibliothek für die Erstellung einer Applikation im Rahmen dieses Projekts machen. Da die Bibliothek für C++, Python und Java geschrieben ist, kann eine von den drei Programmiersprachen ausgewählt werden.

Die Stackoverflow Survey 2021 nennt Python, mit Abstand gefolgt von Java und C++, als die beliebteste Sprache unter den dreien. Auch Pythons Data Science Bibliotheken wie NumPy und Pandas sind bei den Entwicklern sehr beliebt. Während Python von 67,83% der Entwickler gern genutzt wird, sind es bei C++ nur 49,24% und bei Java nur 47,11%. Die Anzahl an Repositories auf GitHub kann dies teilweise bestätigen. Ungefähr 1,7 Millionen Repositories nutzen Python als Sprache. Java kommt auf 1,3 Millionen, während C++ nur in 260 000 Repositories eingesetzt wird. Die Vergleichsaspekte werden in der Tabelle 5.2 nochmals dargestellt. Die Beliebtheit von Python lässt schließen, dass in mehr neuen Projekten mit Python anstatt C++ oder Java gearbeitet wird. [35] Für Python sprechen

Sprache	Nutzung (Stackoverflow)	Beliebtheit (Stackoverflow)	Anzahl Repositories (Github)
Python	48,24%	67,83%	1 786 097
Java	35,35%	47,11%	1 324 561
C++	24,31%	49,24%	259 961

Tabelle 5.2: Vergleich der Sprachen Java, Python und C++

außerdem die Benutzerfreundlichkeit der Sprache, vorhandene Vorerfahrung der Autorinnen des Projekts und die Möglichkeit, zusätzlich scikit-image einzusetzen. Der letzte Faktor wird durch die Kompatibilität von OpenCV und NumPy verstärkt.

Algorithmen für die Bildverarbeitung sind für die zu entwickelnde Anwendung ein wichtiges Thema, denn möglichst effiziente und effektive Algorithmen sollen für die Erstellung des Programms gefunden werden. Deshalb werden sie im nächsten Kapitel 6 genau untersucht und verglichen.

6 Analyse der Algorithmen

Bildverarbeitung ist ein komplexer mehrstufiger Prozess, bei dem große Datenmengen bewältigt werden müssen. Um diese Aufgabe effizient zu lösen, kommen in jeder Stufe unterschiedliche Algorithmen zum Einsatz. In diesem Kapitel werden daher für die verschiedenen Prozesse mögliche, passende Algorithmen untersucht, um einen Überblick über die verfügbaren Methoden zu erhalten. Für den geplanten Bildverarbeitungsprozess sind Algorithmen zum Einlesen der Bilder, für Merkmalerkennung, Texturenanalyse und Bildreparatur denkbar. Diese dienen als Basis für die spätere Implementierung der Anwendung und werden in diesem Kapitel in der genannten Reihenfolge untersucht.

6.1 Algorithmen zum Einlesen mehrerer Bilder

Um mehrere Bilder aus unterschiedlichen Unterordnern automatisch einzulesen, muss der relative Pfad aller Bilder angegeben werden. Da die Unterordner sowie die Bilddateien selbst unterschiedliche Bezeichner haben, müssen diese mithilfe sogenannter Wildcards angegeben werden. Danach wird durch alle Pfade durchiteriert und dabei jede Datei eingelesen. Hier kommen neben Methoden der Bildverarbeitung auch Methoden für die Dateiverwaltung zum Einsatz. [36] [37]

6.2 Algorithmen für Merkmalerkennung (Feature Detection)

Die Erkennung von Bildmustern basiert auf den Merkmalen des Objekts, und diese Merkmale müssen aus dem Objekt im Bild extrahiert werden. Laut dem Autor Zhang [4] gibt es im Image Engineering mehrere verwandte, aber nicht identische Definitionen von Feature oder Merkmal:

- Schlüsselinformation, die von einer Gruppe von Pixeln getragen wird, um ein Bild oder sein Objekt zu identifizieren.
- Ein Teil des Bildes mit bestimmten Eigenschaften. Typische Beispiele sind Kanten, Objekt Konturen und texturierte Bereiche in einem Bild.
- Einige Ereignisse mit charakteristischen Merkmalen (zum Beispiel, Augen im Gesicht) oder Attributen die vom Objekt abgeleitet sind (zum Beispiel, Kreisform).
- Kann die Menge der spektralen Merkmale des Objekts im Bild ausdrücken.
- Messergebnisse von Objekten im Bild, die bei der Objektklassifizierung helfen können.
- Ein numerisches Attribut (das auch mit anderen Attributen kombiniert werden kann, um einen Merkmalsvektor zu erzeugen), das üblicherweise in Klassifikatoren verwendet wird.

Erkennung von Merkmalen (Feature Detection oder Feature Location) ist die Identifizierung eines bestimmten Merkmals in einem Bild oder einem Modell. Üblich sind die Algorithmen für Linien-, Kanten-, Grat-, Eckenerkennung, Blob-Analyse und SIFT. [4]

6.2.1 Segmentierungsalgorithmen

Die Bildsegmentierung gehört zum Teilbereich der Bildanalyse. Oft liegt der Fokus bei der Verwendung und Analyse von Bildern auf dem Vordergrund oder einem speziellen Objekt. Diese Objekte ergeben die Region Of Interest (ROI). Bei anschließenden Analysen z.B. zur Gesichtserkennung wird nicht mehr das ganze Bild analysiert, sondern nur noch die ROI, also das Gesicht in diesem Beispiel. Mithilfe von Segmentierungsalgorithmen können Bilder in Abschnitte geteilt werden, um die Abschnitte von Interesse zu extrahieren. Abschnitte umfassen eine Anzahl an Pixel, die ein sinnvolles Objekt darstellen. Charakteristiken für die Segmentierung sind beispielsweise Farbe, Textur, ähnliche Feature-Orientierung, Feature-Bewegung oder Form der Oberfläche.

Die Segmentierung erstellt eine abstrahierte, kompaktere Version des Originalbildes, wodurch dieses auf einer höheren Ebene analysiert werden kann. Abbildung 6.1[38] und Abbildung 6.2 [38] zeigen ein Originalbild und das gleiche Bild nach einer darauf angewandten Segmentierung. In diesem Beispiel soll durch Segmentierung die Fahrbahnmarkierung erkannt werden. [4]



Abbildung 6.1: Originalbild

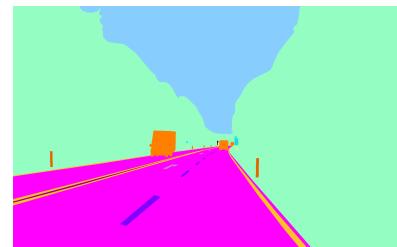


Abbildung 6.2: Bild nach Segmentierung

Einige Techniken der Bildsegmentierung nutzen die Untersuchung des Graulevels zur Segmentierung. Die Pixel eines Bildes können sequentiell oder parallel analysiert werden. Sequentielle Techniken können Ergebnisse aus vorangegangenen Pixelanalysen am selben Bild bei der weiteren Analyse nutzen, während parallele Verfahren gleichzeitig und unabhängig von den vorher analysierten Pixeln ausgeführt werden. Im Folgenden werden einige dieser Techniken näher beschrieben.

Regionbasierte Bildsegmentierung

Bei der regionbasierten Bildsegmentierung wird eine bestimmte Anzahl an Bildregionen erstellt basierend auf einem Homogenitätskriterium. Kriterien sind z.B. Helligkeit oder Distanz, die sich auf das Bild in Form von Farbe, Textur, Form oder Krümmung auswirken. Um Regionen zu finden und zu segmentieren werden häufig Ähnlichkeiten im Graulevel gesucht. Eine Region enthält alle Pixel, die in Anbetracht des Homogenitätskriterium in Verbindung mit ihren Nachbarpixeln stehen. Verschiedene Regionen sind durch Grenzen getrennt. Zwei Pixel

innerhalb einer Region müssen sich an jedem Punkt über Pixel derselben Region miteinander verbinden lassen. Bekannte Methoden sind die Wasserscheidentransformation und das Mean-Shift-Clustering. Algorithmen der farbbasierten Bildsegmentierung nutzen ebenfalls Methoden der regionbasierten Bildsegmentierung. Das Farbbild wird nach Kriterien basierend auf dem Farbspektrum abhängig des Farbraums in Regionen eingeteilt.

Grenzbasierte Bildsegmentierung

Die Grenzbasierte Bildsegmentierung funktioniert gegensätzlich zur regionbasierten Bildsegmentierung: Anstatt eines ähnlichen Graulevels werden Unstetigkeiten im Graulevel gesucht, um die Objektgrenzen zwischen Regionen herauszufinden. Zur Grenze gehören alle Pixel, die nicht zu einer Region zugeordnet sind. Die bekanntesten grenzbasierten Methoden sind die kantenbasierte Segmentierung und gratbasierte Segmentierung. [4] [39]

Es wurden zwei bekannte Verfahren in der Bildsegmentierung beschrieben. Im Folgenden Abschnitt werden Algorithmen vorgestellt, die die oben genannten Techniken anwenden.

6.2.1.1 Wasserscheidentransformation (Watershed)

Bei der Wasserscheidentransformation wird das Prinzip der Wasserscheide zur Ermittlung von Regionen auf Graustufenbilder angewandt. Abbildung 6.3 [40] illustriert die Wasserscheide anhand von Becken, die rechts und links einen mehr oder weniger hohen Rand haben. Der zentrale Punkt eines Becken ist das lokale Minimum. Würde das Becken mit Wasser geflutet werden, würde rechts und links an den höchsten Punkten das Wasser überschwappen. Diese Punkte stellen die Wasserscheiden-Linien dar. Die geflutete Fläche bildet eine Region, die Wasserscheiden-Linien stellen die Grenze dar.

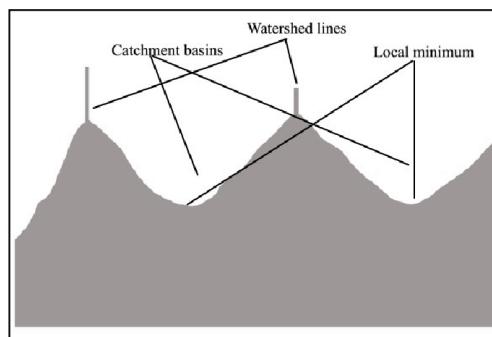


Abbildung 6.3: Illustration einer Wasserscheide

Für die Wasserscheidentransformation wird die mathematische Morphologie genutzt. Mithilfe des Graulevels eines Bildes kann eine topographische Sicht auf das Bild erstellt werden, um lokale Minima zu berechnen. Die Grenzen der Regionen werden mithilfe der Regionen selbst festgelegt, der Algorithmus ist daher regionbasiert und wird sequentiell ausgeführt. Die Wasserscheidentransformation lässt sich in 3 Schritte unterteilen:

1. Kanten innerhalb des Bildes finden

2. Distanz Transformation der Kante berechnen
3. Region der Wasserscheide der Distanz Transformation berechnen

Anhand der Abbildung 6.4 [41] können die einzelnen Schritte einer typischen Implementierung der Wasserscheidentransformation verfolgt werden. Im ersten Schritt werden alle Kanten im Bild gesucht. Für Objekte wie im linken Bild, deren Grenzen sich überlappen, kommt die Wasserscheidentransformation zum Einsatz. Dazu wird auf der topographischen Ansicht die Distanz Transformation berechnet. Dies ist eine Funktion, die vom lokalen Minimum den nächstgelegenen Grenzpunkt berechnet, wie im mittleren Bild zu sehen ist. Jedes Minima wird eindeutig bezeichnet, um so die späteren Regionen identifizieren zu können. Der Algorithmus ermittelt mittels der Distanz Transformation die Wasserscheiden-Linien. Diese können von der topographischen Ansicht auf das Bild übertragen werden. Das rechte Bild zeigt das Ergebnis der Segmentierung. [4]

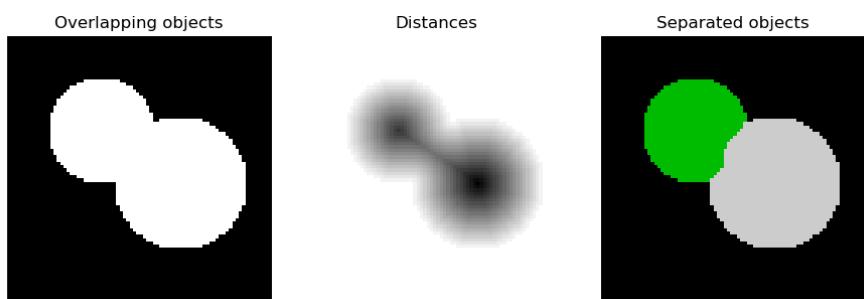


Abbildung 6.4: Anwendung der Wasserscheidentransformation

6.2.1.2 Mean-Shift-Clustering

Das Mean-Shift-Clustering wendet Techniken des Mean-Shift zur Segmentierung eines Bildes an. Der Mean-Shift basiert auf dem Anstieg des Dichtegradienten und wird zur Bestimmung komplexer Feature Cluster verwendet. An einem Punkt wird die durchschnittliche Abweichung berechnet, um danach den Punkt zu der durchschnittlichen Abweichung zu verschieben. Ziel ist es, die Punkte eines Clusters in die Mitte des Clusters zu verschieben, wo die Dichte am höchsten ist. Die Abweichung der Dichte der Punkte zeigen daher immer in die Richtung des Gradienten.

Abbildung 6.5 [42] zeigt die Bewegung der Punkte in Richtung Gradient in zeitlichen Abständen. Links im Bild sind die Punkte an ihrer Originalposition. Im mittleren Bild bewegen sie sich Richtung Gradienten, im rechten Bild haben sie den Gradienten erreicht.

Abbildung 6.6 [43] zeigt eine Anwendung des Mean-Shift-Clustering an einem Bild. Das obere Bild wurde anhand der Farben in Regionen aufgeteilt. Im unteren Bild nach dem Clustering hat jede Region den jeweiligen durchschnittlichen Farbwert der Region erhalten. Dadurch wurde die Farbwertverteilung stark verringert. [4]

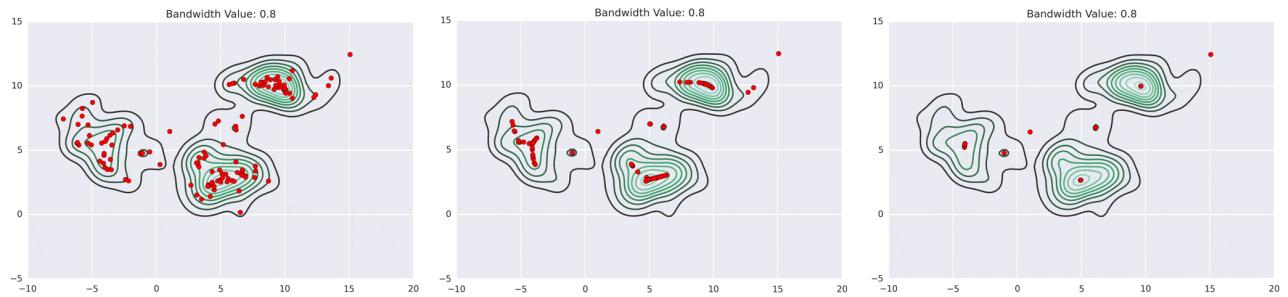


Abbildung 6.5: Vorgehen beim Mean-Shift-Clustering

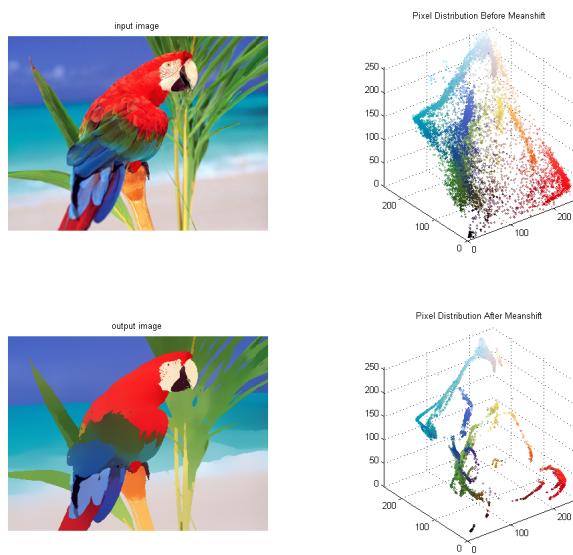


Abbildung 6.6: Anwendung des Mean-Shift-Clustering an einem Bild

6.2.1.3 Schwellenwertverfahren (Thresholding)

Die Schwellenwertbildung (Thresholding) ist ein regionenbasierter paralleler Algorithmus zur Bildsegmentierung. Diese Art von Algorithmus ist die am weitesten verbreitete, parallele, direkte Segmentierungsmethode zur Erkennung von Regionen, auch bekannt als Parallel-Region-Technik. Andere ähnliche Methoden, wie die Klassifizierung im Merkmalsraum der Pixel, können als seine Verallgemeinerung angesehen werden. Das Hauptproblem dieser Methodengruppe besteht darin, einen geeigneten Schwellenwert zu wählen. Nachdem der Schwellenwert festgelegt ist, ist die Segmentierungsmethode für 2D- und 3D-Bilder gleich. Der Pixel- oder Voxelwert (Voxel ist ein dreidimensionales Äquivalent eines Pixels) wird mit dem Schwellenwert verglichen und seine Zugehörigkeit wird bestimmt.

Die einfachsten Schritte zur Verwendung der Schwellenwertmethode zur Segmentierung eines Graustufenbildes sind folgende. Zunächst wird eine Grauschwelle T ($g_{\min} < T < g_{\max}$) für ein Bild mit einem Grauwert zwischen g_{\min} und g_{\max} bestimmt. Dann wird der Grauwert jedes Pixels im Bild mit dem Schwellenwert T verglichen, und entsprechend den Vergleichsergebnissen werden die Pixel in zwei

Kategorien eingeteilt: diejenigen mit einem Grauwert größer als der Schwellenwert sind eine Kategorie, und diejenigen mit einem Grauwert kleiner als der Schwellenwert sind eine andere Kategorie (Pixel mit einem Grauwert gleich dem Schwellenwert können in eine dieser beiden Kategorien eingeordnet werden). Diese beiden Arten von Pixeln entsprechen im Allgemeinen den Vordergrund- und Hintergrundregionen im Bild, so dass die Schwellenwertberechnung dazu verwendet werden kann, das Bild zu segmentieren. Die oben genannte Segmentierungsmethode mit nur einem Schwellenwert wird als Single-Thresholding-Methode bezeichnet, während die Methode, die mehrere Schwellenwerte verwendet, als Multi-Thresholding-Methode bezeichnet wird. [4]

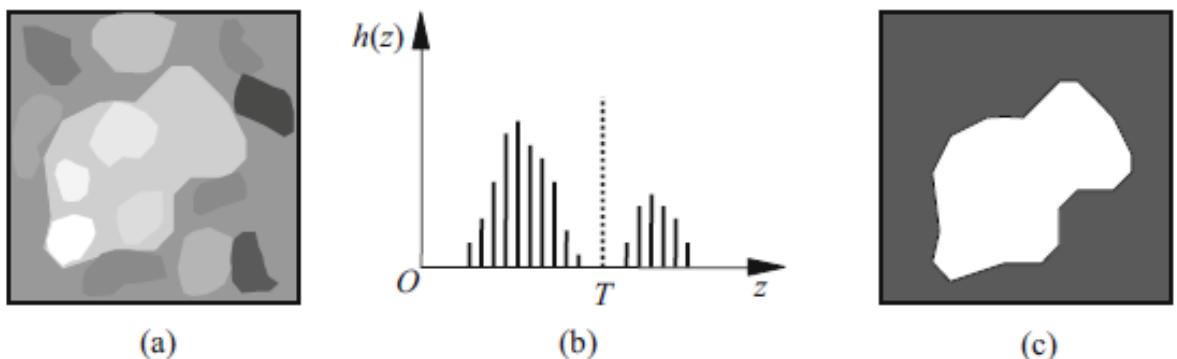


Abbildung 6.7: Schematische Darstellung von Thresholding

Die Schwellenwertbildung quantisiert ein Graustufenbild in ein Binärbild. Abbildung 6.7 [4] zeigt ein Beispiel mit einer einzigen Schwelle. In (a) kann man sehen, dass unterschiedliche Grauwerte in mehreren Regionen enthalten sind; Abbildung (b) zeigt ein Histogramm zum Bild, wobei z der Grauwert des Bildes ist und T der Schwellenwert. Abbildung (c) stellt das Segmentierungsergebnis dar, wobei Pixel, die größer als der Schwellenwert sind, weiß dargestellt werden, und diese, die kleiner sind, sind schwarz.

Bei der Schwellenwertbildung ist die Bestimmung des Schwellenwerts der Schlüssel. Je nach unterschiedlichen Grundlagen für die Schwellenwertbildung können Schwellenwerte in pixelabhängige, regionabhängige und koordinatenabhängige Schwellenwerte unterteilt werden. Auf dieser Grundlage kann die Schwellenwerttechnologie in drei Arten von Technologien unterteilt werden, die diese drei Arten von Schwellenwerten anwenden.

Beim **pixelabhängigen** Threshold wird nur die Art der einzelnen Pixel für der Auswahl berücksichtigt. Er wird auch globaler Schwellenwert genannt, weil der zu diesem Zeitpunkt ermittelte Schwellenwert die globale Beschaffenheit des Bildes benutzt und für das gesamte Bild gilt. Bei der Auswahl der **regionabhängigen** Schwellenwerte muss nicht nur die Beschaffenheit jedes Pixels selbst, sondern auch der lokalen Region (Nachbarschaft) jedes Pixels berücksichtigt werden. Diese Art von Schwell-

lenwerten wird auch als lokaler Schwellenwert bezeichnet, da er die lokalen Eigenschaften des Bildes nutzt. Um einen **koordinatenabhängigen** Schwellenwert zu finden, sind nicht nur die Eigenschaften der einzelnen Pixel selbst und der lokalen Region (Nachbarschaft) jedes Pixels notwendig, sondern auch die Koordinaten der Position jedes Pixels im Bild. Dieser Schwellenwert wird auch dynamisch genannt, weil er sich Punkt für Punkt im Bild ändert, das heißt er hängt von den Pixelkoordinaten ab. Mit anderen Worten: Für jede Koordinatenposition im Bild wird ein Schwellenwert benötigt. [4]

Der springende Punkt bei allen Schwellenwertverfahren ist die Wahl eines geeigneten Schwellenwertes. Dieser kann durch einen menschlichen Bearbeiter sinnvoll gewählt werden. Da beim lokalen und dynamischen Schwellenwertverfahren aber eine größere Anzahl an Schwellenwerten benötigt wird, bietet es sich an, ein automatisches Verfahren zur Bestimmung der Schwellenwerte einzusetzen. Es gibt eine große Anzahl an konkreten Verfahren zur Wahl eines geeigneten Schwellenwertes.

Sowohl zur manuellen als auch zur automatischen Festlegung eines Schwellenwertes ist das Histogramm das wichtigste Hilfsmittel. Lokale Maxima weisen auf die Grauwerte oder Grauwertbereiche hin, die im Bild vom Hintergrund oder von größeren Objekten angenommen werden. Im Idealfall ist das Histogramm bimodal, das heißt, es lassen sich zwei deutlich voneinander getrennte Maxima erkennen. Ein einfacher, aber auch fehleranfälliger Ansatz ist es, den Mittelwert zwischen den beiden Maxima als Schwellenwert zu wählen. Eine weitere, einfache Herangehensweise ist es, den Grauwert des Minimums zwischen den Maxima als Schwellenwert festzulegen. Eines der anspruchsvollen Verfahren zur automatischen Bestimmung von Schwellenwerten ist das Verfahren von Otsu, das sich als Standard etabliert hat. Es wurde in scikit-image sowie in OpenCV implementiert. [44]

Otsu Verfahren. Nobuyuki Otsu hat im Jahr 1979 eine neue Methode für die Bestimmung des Schwellenwerts vorgeschlagen. Die Methode zur automatischen Auswahl eines Schwellenwerts aus einem Graustufenhistogramm wurde aus der Sicht der Diskriminanzanalyse abgeleitet. Diese befasst sich direkt mit dem Problem der Bewertung der Güte von Schwellenwerten. Ein optimaler Schwellenwert (oder eine Reihe von Schwellenwerten) wird nach dem Diskriminanzkriterium ausgewählt, das heißt durch Maximierung des Diskriminanzmaßes (oder des Maßes der Trennbarkeit der resultierenden Klassen in Graustufen). Die Methode zeichnet sich durch ihre nichtparametrische und nicht überwachte Schwellenwertauswahl aus und hat laut Otsu folgende Vorteile:

- Das Verfahren ist sehr einfach.
- Eine unkomplizierte Erweiterung auf Multithresholding-Probleme ist möglich.
- Ein optimaler Schwellenwert (oder eine Reihe von Schwellenwerten) wird automatisch und stabil ausgewählt, nicht auf der Grundlage der Differenzierung (das heißt einer lokalen Eigenschaft wie zum Beispiel Tal), sondern auf der Integration (einer globalen Eigenschaft) des Histogramms. [11]

Der Algorithmus kann wie folgt dargestellt werden. Für jeden potenziellen Schwellenwert T:

- Werden die Pixel in zwei Cluster entsprechend dem Schwellenwert getrennt.
- Wird der Mittelwert jedes Clusters ermittelt.
- Wird die Differenz zwischen den Mittelwerten quadriert.
- Wird die Anzahl der Pixel in einem Cluster mit der Anzahl der Pixel im anderen Cluster multipliziert. [45]

6.2.1.4 Canny Algorithmus

Der Canny Operator ist einer der komplexesten Operatoren zur Kantenextraktion. Der Algorithmus wurde auf identische Testbilder angewandt, die für andere Kantenoperatoren verwendet wurden. Der Vergleich zeigt eine gute Leistung des Canny-Algorithmus im Vergleich zu Sobel-Operator und Laplacian of Gaussian (LoG), wie in der Abbildung 6.8 [10] dargestellt. Der Operator, der die Gradien-

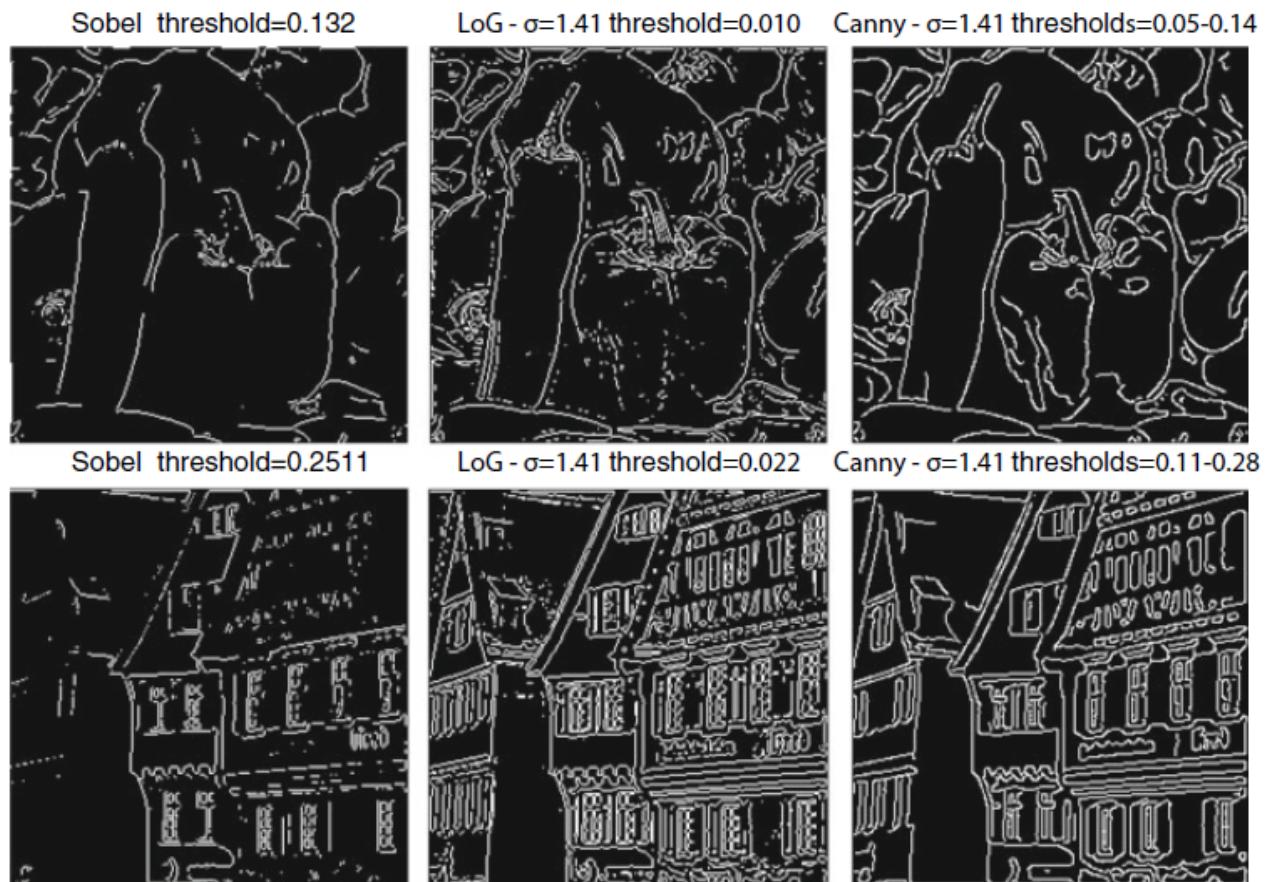


Abbildung 6.8: Canny-Algorithmus im Vergleich zu LoG und Sobel-Operator

tenmessung approximiert, um die Kanten zu extrahieren, muss zwei gegensätzliche Eigenschaften lösen:

- Das Rauschen abschwächen;
- die Kanten so genau wie möglich zu lokalisieren.

Der Canny-Operator erwies sich als optimal für die Lösung des Komromisses zwischen genauer Kantenlokalisierung und Rauscheinfluss. Der Operator kann wie folgt charakterisiert werden:

1. *Gute Kantenerkennung.* Das Signal-Rausch-Verhältnis (Signal-Noise Ratio (SNR)) des Gradienten wird maximiert um die geringste Fehlerwahrscheinlichkeit bei der Bestimmung einer echten Kante zu erhalten und eine minimale Wahrscheinlichkeit, eine falsche Kante als gut zu erkennen.
2. *Gute Lokalisierung.* Punkte, die als Kanten identifiziert werden, sind so nah wie möglich an der Mitte der tatsächlichen Kante.
3. *Einzelne Antwort.* Der Operator sollte eine einzige Antwort für dieselbe Kante geben. [10]

Die Erkennung von Kanten umfasst vier Schritte (in der Abb. 6.9 [46] ist der Ablauf in 6 Schritten dargestellt):

1. Gaußsche Glättung des Bildes, um Rauschen zu reduzieren und kleine Details zu eliminieren;
2. Berechnung Gradientenintensität und -richtung jedes Pixels;
3. Verwendung von Nicht-Maximum Unterdrückungsmethoden, um kleine Gradienten zu eliminieren und sich auf die Kantenpositionierung zu konzentrieren;
4. Den Schwellenwert des Gradienten nehmen und die Kantenpunkte verbinden. Hier wird die Hysterese verwendet: Zuerst werden die starken Kantenpixel verbunden und dann die schwachen.[4]

Der Canny-Algorithmus ist robuster bei der Erkennung von Bilddetails durch bessere Lokalisierung der Kanten und Minimierung der Artefakte (filtert Rauschen besser). Er benötigt keine Ausdünnung der erkannten Kanten, auch wenn es mehrere Kalibrierungsschritte erfordert, um die optimalen Schwellenwerte zu finden. [10]

Der Algorithmus ist in scikit-image und in OpenCV implementiert.

6.2.1.5 Laplace Filter

Der Laplace-Filter nutzt für die Kantenerkennung die zweite Ableitung eines Bildes. Er basiert auf dem *Gradientenfilter*, der als Kantenfilter Diskontinuitäten der Intensitätswerte hervorhebt und die niedrigen Frequenzen eliminiert. Dadurch bleiben nur noch große Veränderung in der Intensität zurück. Abbildung 6.10 [47] zeigt die Intensitätsfunktion in Abhängigkeit zur Bildposition mit einer hohen Diskontinuität, die als Kante erfasst wird.

Zur Hervorhebung der Diskontinuitäten eignet sich die erste Ableitung. Abbildung 6.11 [10] veranschaulicht die Diskontinuität einer Funktion $f(x)$ und die dazugehörige erste Ableitung. Der Punkt

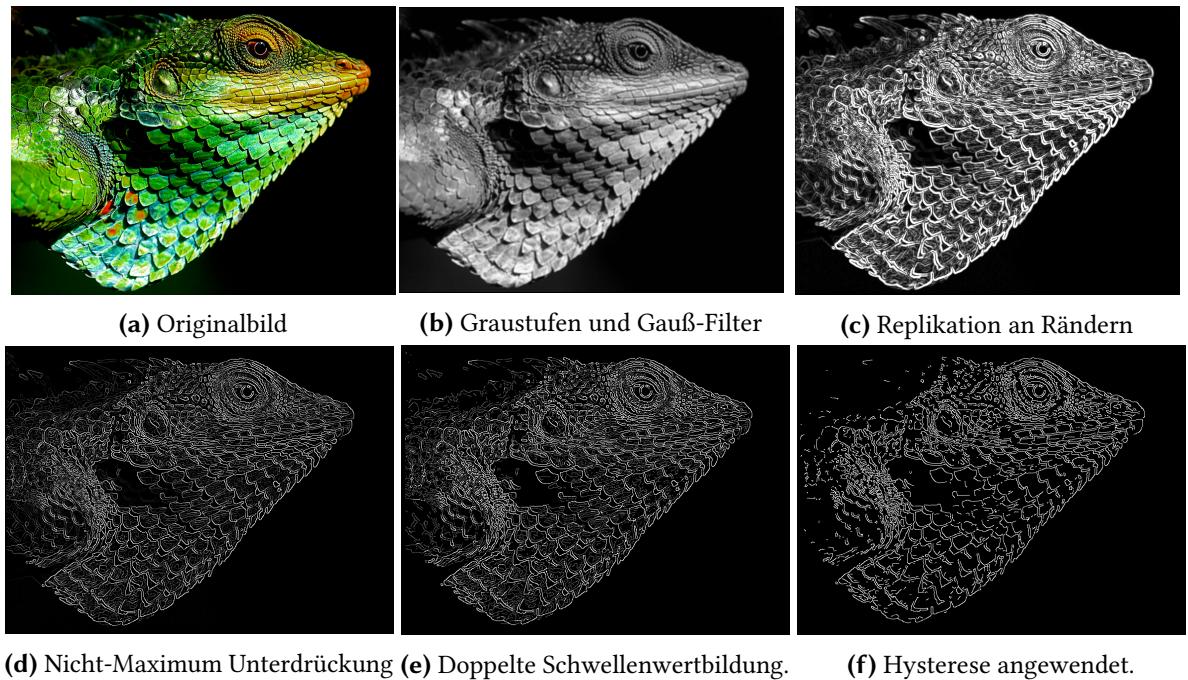


Abbildung 6.9: Schrittweise Kantenerkennung eines Bildes durch Canny Algorithmus

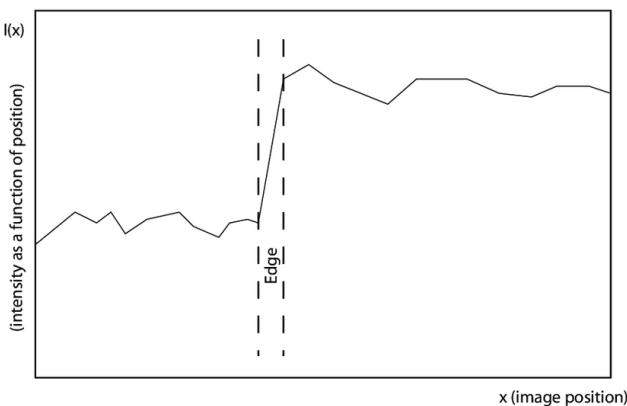


Abbildung 6.10: Erkennung der Kante anhand der Intensitätsfunktion

der maximalen Diskontinuität, also der Punkt der höchsten Steigung, wird in der ersten Ableitung als Maximum bzw. Minimum dargestellt.

Um die Richtung der Kante eines zweidimensionalen Bildes zu erkennen, müssen die verschiedenen Richtungen, in die die Kante verlaufen kann, in Betracht gezogen werden. Dazu wird der maximale Wert des Richtungsgradienten anstelle der Extremwerte der ersten Ableitung genutzt.

Der Laplace-Filter wird wie der Gradientenfilter auf Graustufenbilder angewandt. Diese stellen die Intensität in den verschiedenen Grautönen dar. Wie Abbildung 6.12 [10] zeigt, wird in der zweiten Ableitung kein Maximum, sondern ein Nulldurchgang als Erkennungsmerkmal für Kanten genutzt.

Der Laplace-Operator als zweidimensionale Funktion ist definiert als die Summe der zweiten Ablei-

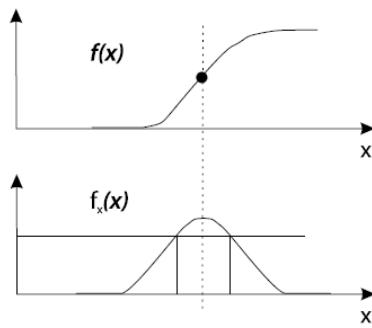


Abbildung 6.11: Wechsel der Diskontinuität

tungen einer Funktion in Richtung der x - und y -Achsen. Im Gegensatz zum Gradientenfilter, wo eine Schwelle über bzw. unter dem Extremwert die Breite der Kante festlegt, sind Kanten bei Untersuchung mit dem Laplace-Filte typischerweise 1 Pixel breit. Grund dafür ist die exaktere Feststellung von Kanten unter dem Laplace-Filter. Dafür ist dieser anfälliger für Rauschen, da der Nulldurchgang präzise lokalisiert werden muss. Es werden auch keine Informationen zur Richtung der Kante gegeben, daher kann es passieren, dass doppelte Kanten erkannt werden, die nebeneinander sind. [10]

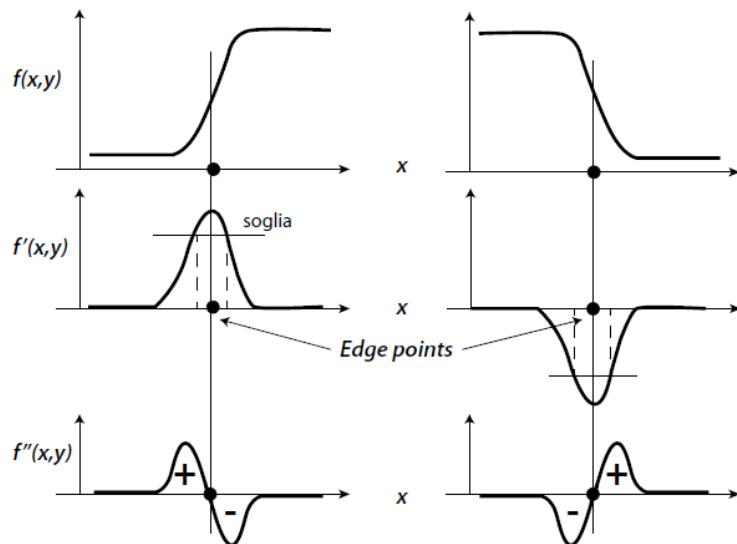


Abbildung 6.12: Zweite Ableitung zur Erkennung von Kanten

Wie der Canny Algorithmus ist auch der Laplace Filter in den Bibliotheken scikit-image und OpenCV implementiert.

6.2.2 Algorithmen für die Texturenanalyse

Textur ist eine Charakteristik, die wichtig zur Beschreibung des Bildinhaltes ist und daher häufig in der Bildanalyse angewandt wird. Die Texturenanalyse beinhaltet die Beschreibung, Repräsentation, Segmentierung, Klassifikation und Synthese von Texturen. Die Oberfläche ist Teil der Objekte, die sich dadurch identifizieren lassen. Durch Textur werden Eigenschaften der räumlichen Verteilung des

Graulevels wie z.B. Ebenmäßigkeit oder Gleichmäßigkeit beschrieben. Farbtexturen können durch Eigenschaften der Farbe, Reflektierung oder Beleuchtung bestimmt werden. Textur zeigt sich als Muster, das durch vergrößern der Objekte eines Bildes sichtbar wird. Sie ist ein Maß für die Beziehung zwischen Pixeln einer lokalen Region.

Texturelemente sind kleine Einheiten, die aus geometrischen Formen bestehen. Diese Elemente wiederholen sich in der Textur und bilden so Charakteristiken in der Textur. Ist der Maßstab der Texturelemente klein in Relation zum Abstand der Elemente, spricht man von Mikrotextur. Ist das Gegenteil der Fall, so haben die Texturelemente Formen, die größer als der Abstand zwischen den Elementen ist. Dies wird der Makrotextur zugeordnet. In der Texturenanalyse werden Methoden aus den folgenden drei Kategorien angewandt:

- Statistischer Ansatz: Messen der räumlichen Verteilung der Pixelwerte.
- Struktureller Ansatz: Beschreibung der Textur durch Strukturinformationen der Strukturelemente.
- Spektrum-Ansatz: Beschreibung der globalen Regelmäßigkeiten der Textur mithilfe der spektralen Verteilung. [4]

Im Folgenden werden bekannte Methoden der Texturenanalyse beschrieben.

6.2.2.1 Binary Gabor Pattern

Das Binary Gabor Pattern (BGP) ist ein Muster zur Repräsentation von Texturen. Es basiert auf dem Local Binary Pattern (LBP). Die Theorie dahinter ist, dass ein Bild aus mehreren Textureinheiten besteht, die das Texturspektrum des Bildes ausmachen. Eine Textureinheit bildet sich aus 8 Elementen, die die Werte 0,1 oder 2 haben können. Sie sind Teil einer Nachbarschaft bestehend aus 3 x 3 Pixel. Es gibt $3^8=6561$ mögliche Textureinheiten, die die Textur einer Nachbarschaft beschreiben können. Bei Anwendung des BGP wird eine Auswahl an vordefinierten Textureinheiten gespeichert. Eine Ganzzahl soll eine Textureinheit repräsentieren können. Danach kann ein Histogramm erstellt werden, das das Vorkommen der einzelnen Textureinheiten darstellt.

Das LBP verwendet zur Erkennung von Unterschieden zwischen den Pixeln die lokalen Nachbarn, die wiederum in Binärwerte umgewandelt werden. Dadurch ist LBP empfindlich gegenüber Rauschen. Um die Rauschempfindlichkeit zu verbessern, werden im BGP diese Unterschiede nicht zwischen Nachbarpixeln, sondern zwei Regionen berechnet. Dafür kommt der Gabor Filter zum Einsatz. Zusätzlich kann der BGP rotierte Texturen erkennen, indem dieselbe Textureinheiten in rotierten Positionen gespeichert wird. Abbildung 6.13 [12] veranschaulicht den Ablauf des BGP. Zuerst wird der Gabor Filter auf das Bild angewandt, dann die zugeordnete Ganzzahl in binär umgewandelt. Danach kann dem Binärwert eine eindeutige BGP Nummer zugewiesen werden, um die Textureinheit an dieser Stelle zu beschreiben. [12] [48]

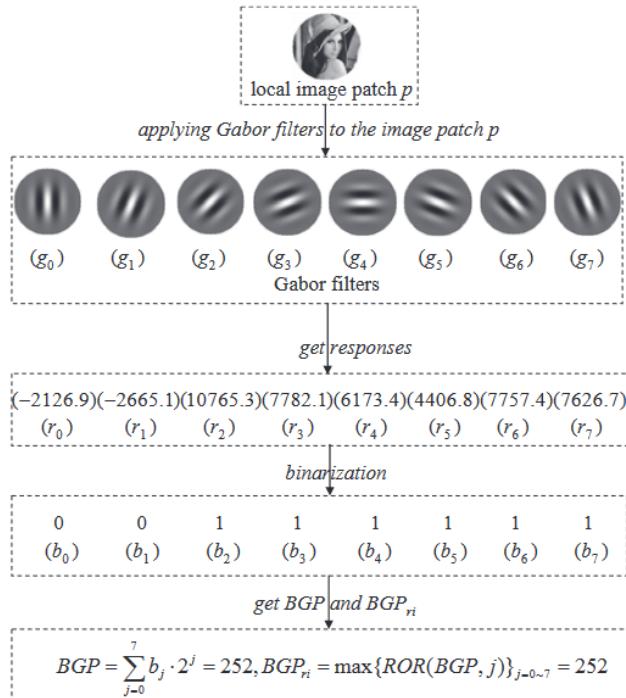


Abbildung 6.13: Darstellung des BGP Prozess

6.2.2.2 Morphologische Filter

Morphologische Filterung (Morphological Filtering) ist eine Filtertechnik, die mathematische morphologische Operationen auf ein Bild anwendet. Solch ein Filter ist ein nichtlinearer Filter, der die geometrischen Merkmale eines Bildes lokal durch morphologische Transformation modifiziert. Betrachtet man jedes Bild im euklidischen Raum als eine Menge, ist die morphologische Filterung eine Mengenoperation, die die Form des Bildes verändert. Ausgehend von der Filterung und dem Ergebnis der Filterung lässt sich eine quantitative Beschreibung der Geometrie des Eingangsbildes erhalten. Eine Implementierung des morphologischen Filters ist eine Kombination aus Öffnen (Opening) und Schließen (Closing). *Opening* und *Closing* sind grundlegende mathematische Morphologieoperation für Bilder. Je nach Art des Bilds, wie Binärbild, Graustufenbild und Farbbild, können sie in Binary Opening/Binary Closing, Gray-Level Opening/Gray-Level Closing und Color Opening/Color Closing unterteilt werden. Der Open Operator ist ein Operator, der die mathematische Morphologieoperation an einem binären Bild durchführt. Der Operator kann aus einer Folge von N-Erodieroperatoren, gefolgt von N-Dilatationsoperatoren, die alle dasselbe spezifische strukturierende Element verwenden, zusammengesetzt werden. Dieser Operator ist sowohl für die Trennung von kontaktierenden Objekten als auch für die Entfernung kleiner Regionen nützlich. [4]

Abbildung 6.14 [49] zeigt das Opening eines Binärbildes mit einem Kreis als strukturierendem Element. Es handelt sich um das nacheinander Ausführen einer Erosion und einer Dilatation jeweils mit demselben strukturierenden Element. Durch die Erosion werden alle Strukturen gelöscht, die

kleiner sind als das strukturierende Element. Die anschließende Dilatation macht die Erosion für den verbleibenden Rest wieder rückgängig. [49]

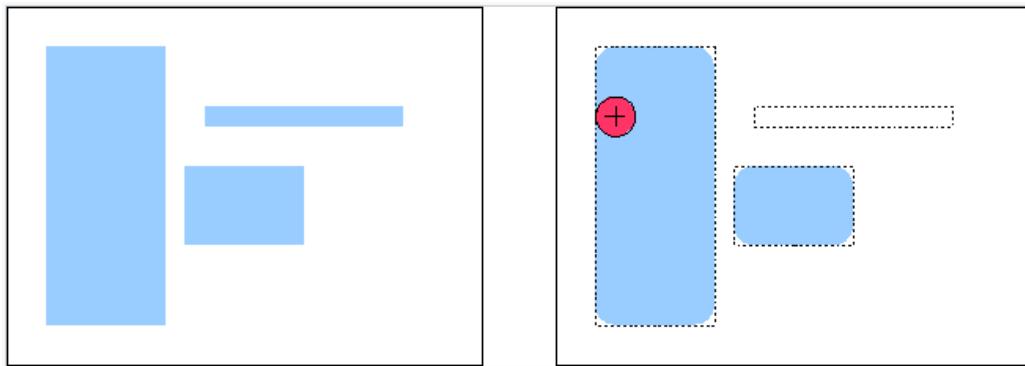


Abbildung 6.14: Der Effekt des Opening Operators

Beim Closing werden zwei binäre morphologische Operationen nacheinander verwendet: zuerst die Dilatationsoperation und dann die Erosionsoperation. Dies führt dazu, dass kleine Löcher oder Lücken im Bild gefüllt werden. [4]

Es handelt sich um das nacheinander Ausführen einer Dilatation und einer Erosion auf das Bild A jeweils mit demselben strukturierenden Element X . Durch die Dilatation werden alle Löcher geschlossen, in die das strukturierende Element nicht vollständig hineinpasst. Die anschließende Erosion reduziert das Bild wieder so weit, dass es möglichst nahe an das Original herankommt. Die durch die Dilatation vollständig geschlossenen Löcher entstehen dabei nicht mehr. Nur teilweise geschlossene Löcher werden wieder aufgeweitet. Ein Beispiel des Closing ist in der Abbildung 6.15 (Bildquelle: [50]). [50]

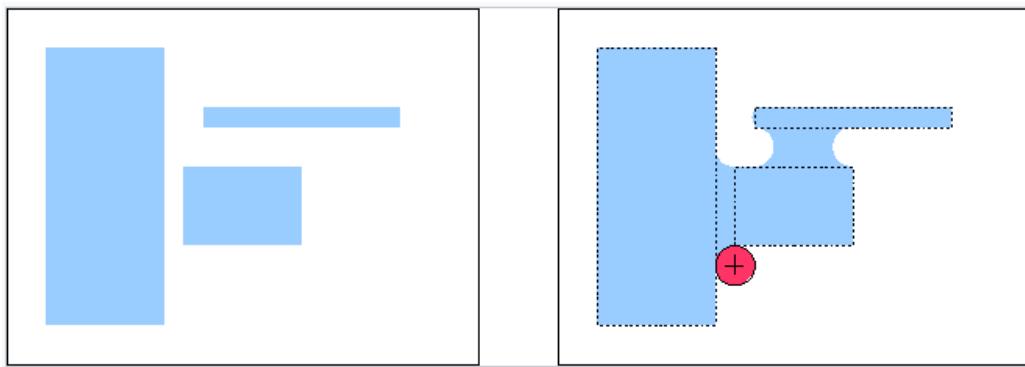


Abbildung 6.15: Der Effekt des Closing Operators

Die Oberflächenzerlegung durch morphologische Verfahren wird als eine Art von Siebtechnik betrachtet, die auf die morphologische Granulometrie zurückgehen könnte. Sie ist vergleichbar mit dem Sieben kleiner fester Partikel mit einer Reihe von Sieben mit zunehmender Maschenöffnung. Das Sieb mit der kleinsten Öffnung wird zuerst verwendet. Die Körner, die größer als die Maschenöffnung sind, werden zurückgehalten und gezählt. Mit dem größeren Sieb werden dann die restlichen Körner

gesiebt, und dieser Vorgang wird fortgesetzt, bis alle Siebe verwendet worden sind. Auf diese Weise werden die Körner nach der Größe der Maschenöffnungen sortiert. [51]

Methoden von mathematischer Morphologie werden in vielen Bildverarbeitungsbibliotheken verwendet, darunter sind scikit-image und OpenCV.

6.2.2.3 Energievariation

Die Textur eines Bildes kann mithilfe der Energievariation entnommen werden. Die Menge an Energie ist ein statistisches Maß, das aussagekräftige Information zur Textur angibt und Features für die Klassifizierung bietet. Zur Bestimmung der Energievariation werden das LBP, die Graulevel-Co-Occurrence Matrix (Grauwertematrix) und die Kantenerkennung genutzt. Abbildung 6.16 [13] beschreibt den Vorgang in einem Flussdiagramm. Am Originalbild werden zunächst die genannten Methoden angewandt. Danach wird von jedem dieser Ergebnisse mittels einer Formel das Energilevel bestimmt. Aus den drei Energileveln entsteht der Vektor F. Die Energilevel werden im nächsten Schritt miteinander verglichen. Dazu wird der Energiewert des Originalbildes berechnet und der Vektor F davon subtrahiert. Es entsteht ein neuer Vektor, der auch als Featurevektor bezeichnet wird. Dadurch kann ein akkurate Featurevektor für die Texturklassifizierung zur Verfügung gestellt werden.

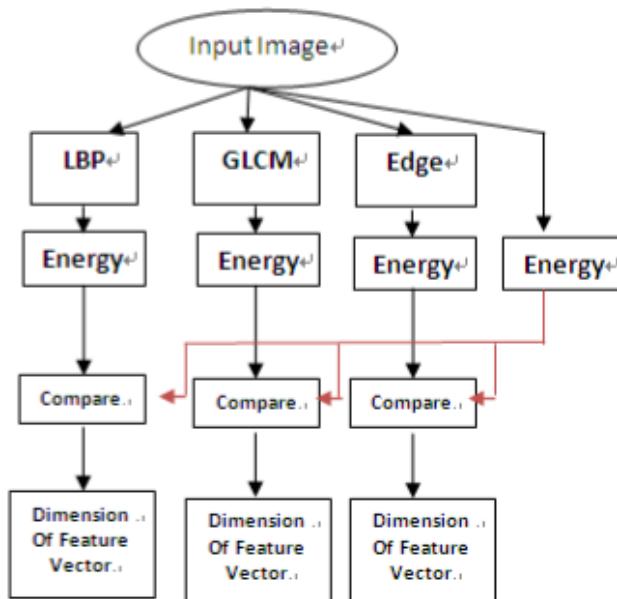


Abbildung 6.16: Ablauf der Energievariation

Diese Vorgehensweise liefert gute Ergebnisse in der Texturklassifizierung mit hoher Genauigkeit und ist daher gut geeignet für die Texturanalyse.

Im folgenden werden die einzelnen Schritte zur Bestimmung der Energiemenge näher beschrieben:

Local Binary Pattern

Das LBP findet Ähnlichkeiten in der Textur mithilfe eines Texturspektrums (mehr dazu in Abschnitt 6.2.2.1).

Grauwertematrix

Die Grauwertematrix ist hilfreich in der Texturanalyse. Mithilfe der Grauwertematrix kann die Statistik zweiter Ordnung in Bezug auf Bildeigenschaften berechnet werden, indem die räumliche Beziehung der Pixel berücksichtigt wird. Die Grauwertematrix zeigt an, wie oft verschiedene Kombinationen von Graustufen in einem Bild vorkommen. Zur Erstellung dieser wird berechnet, wie oft ein Pixel mit dem Intensitätswert i in einer bestimmten räumlichen Beziehung zu einem Pixel mit dem Wert j steht. Die räumliche Beziehung ist standardmäßig zwischen einem Pixel und seinem direkten rechten Nachbar, kann aber auch anders festgelegt werden. Abbildung 6.17 [52] zeigt die Berechnung der Felder einer Grauwertematrix anhand eines Beispielbildes. Der linke Teil der Abbildung zeigt das Originalbild mit der Größe 4x4. Die Nachbarbeziehung 0 - 0 kommt zweimal im Originalbild vor, deshalb wird im rechten Teil in der Matrix in der Zelle oben links eine zwei eingetragen. Diese Zelle steht für die Spalte 0 und für die Zeile 0. Die Beziehung 2 - 3 kommt nur einmal vor, daher wird in der vierten Spalte in der dritten Zeile eine eins eingetragen.

0	1	0	2
0	2	1	1
3	1	0	0
0	0	2	3

4 x 4 image

2	1	3	0
2	1	0	0
0	1	0	1
0	1	0	0

GLCM

Abbildung 6.17: Erstellung der Grauwertematrix

Kantenerkennung

Die Kantenerkennung wird zur Entdeckung von Helligkeitsübergänge in Bildern verwendet. Methoden der Kantenentdeckung basieren häufig auf der ersten oder zweiten Ableitung, wie zum Beispiel der Laplace-Filter (siehe Abschnitt 6.2.1.5). Bei der Einführung der Energievariation wurde der Sobel Operator bevorzugt. [13]

6.2.2.4 Eigenfilter

Eigenfilter sind eine Klasse von adaptiven Filtern. Die meisten für die Texturanalyse verwendeten Filter sind nicht adaptiv, das heißt, die Filter sind vordefiniert und nicht direkt mit der Textur verbunden. Der Eigenfilter bildet jedoch eine Ausnahme, da er datenabhängig ist und die Merkmale der Texturvorherrschaft verstärken kann, weshalb er als adaptiv gilt. Solche Filter werden häufig mit Hilfe

der Karhunen-Loëve-Transformation entwickelt und können auch aus Autokorrelationsfunktionen extrahiert werden. [4]

Die Eigenfilter-Methode bietet nachweislich mehrere Vorteile gegenüber anderen traditionellen Filterentwurfsmethoden. Im Gegensatz zur Methode der kleinsten Quadrate, die die Berechnung einer inversen Matrix erfordert, die für numerische Ungenauigkeiten anfällig sein kann, hat die Eigenfiltermethode eine viel geringere Entwurfskomplexität und bleibt auch dann robust, wenn schlecht konditionierte Matrizen im Entwurfsproblem vorhanden sind. Im Gegensatz zum McClellan-Parks-Algorithmus, der für bestimmte Entwurfskriterien nur schwer modifiziert werden kann, lässt sich die Eigenfiltermethode leicht modifizieren, um eine Vielzahl von Entwurfsbedingungen zu erfüllen. Diese Vorteile, in Verbindung mit der guten Leistungsfähigkeit der Eigenfilter Methode, machen sie zu einer attraktiven Methode für den Filterentwurf. [14]

6.3 Algorithmen für Bildreparatur (Image Restoration)

Die Bildwiederherstellung ist eine große Klasse von Techniken in der Bildverarbeitung. Die Technologie besteht darin, den Prozess der Bildverschlechterung zu modellieren und das Bild entsprechend dem Modell wiederherzustellen. Bezuglich des Ziels des Projekts können solche Techniken nützlich sein, daher werden sie im Folgenden untersucht.

Technologien der Bildrestaurierung können in zwei Kategorien unterteilt werden: die uneingeschränkte und die eingeschränkte Restaurierung. Sie kann entweder automatisch oder interaktiv ablaufen, je nachdem, ob ein externer Eingriff erforderlich ist. Es gibt viele entsprechende Technologien für verschiedene Kategorien. Eine perfekte Bildwiederherstellung ist nur möglich, wenn die degenerierte Funktion mathematisch negiert wird. Typische Bildwiederherstellungstechniken sind die inverse Filterung, die Wiener Filterung und die Wiederherstellung nach der Methode der kleinsten Quadrate. Die *uneingeschränkte* Restaurierungsmethode betrachtet das Bild nur als digitale Matrix und führt die Restaurierung aus einer mathematischen Perspektive durch, ohne die physikalischen Bedingungen, denen das Bild nach der Restaurierung unterworfen werden sollte, zu berücksichtigen. Die Methode der *eingeschränkten* Restaurierung muss auch berücksichtigen, dass das wiederhergestellte Bild bestimmten physikalischen Beschränkungen unterworfen sein sollte. [4]

6.3.1 Inverse Filterung

Das ist eine uneingeschränkte Restaurierungsmethode. Es ist auch die einfachste Bildentschärfungstechnik, die im Allgemeinen im Frequenzbereich arbeitet. [4]

Ein typisches Beispiel von Bildverzerrungen ist eine Verzerrung durch die *Point Spread Function (PSF)* (Abbildung 6.18 [53]). Das Original (Szene) wird mit der PSF gefaltet.

Durch die multiplikative Verknüpfung im Frequenzbereich scheint es nun möglich zu sein, eine verzerrende lineare Filterung durch ein System mit dazu inversem Frequenzgang rückgängig zu

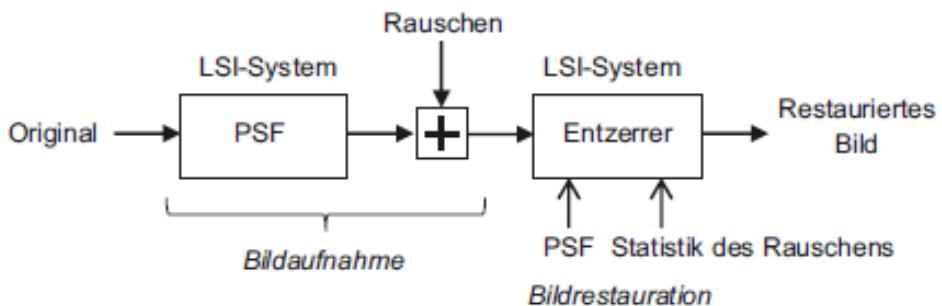


Abbildung 6.18: Lineare Verzerrung und Rauschstörung bei der Bildaufnahme und Bildrestoration durch einen Entzerrer

machen. Der Frequenzgang des Entzerrers ist invers zum Frequenzgang des verzerrenden LSI-Systems. Man spricht auch vom inversen Filter („inverse filtering“) oder der Entfaltung („deconvolution“). Einmal eliminierte Frequenzkomponenten können durch lineare Filterung nicht wieder rekonstruiert werden. Das Original lässt sich nur insoweit restaurieren, als der Entzerrer durch ein inverses Filter dargestellt werden kann. [53]

Der Vorteil des inversen Filters besteht darin, dass er nur die Unschärfe-PSF als Vorwissen benötigt und dass er eine perfekte Wiederherstellung ermöglicht, wenn kein Rauschen vorhanden ist. [54]

6.3.2 Wiener Filterung

Um die Rauschempfindlichkeit des inversen Filters zu überwinden, wurde eine Reihe von Restaurationsfiltern entwickelt, die als *Least-Squares-Filter* bezeichnet werden. Die zwei am häufigsten verwendeten Filter sind der Wiener-Filter und die Methode der kleinsten Quadrate. [54]

Der Prozess, bei dem ein Bild durch Unschärfe und Rauschen beeinträchtigt wird, heißt Degradation. Sei f ein klares Bild, das durch eine Degradationsfunktion H und additives Rauschen n verschlechtert wird, so dass ein verrausches Bild g entsteht:

$$g = f * H + n$$

Der Wiener Filter erfordert statistische Rauschcharakteristika und eine Degradationsfunktion für die Bildwiederherstellung, wobei Rauschen und Bild als unkorreliert angenommen werden. Das Ziel dieses Filters ist es, die Schätzung des klaren Bildes f zu finden, um den mittleren quadratischen Fehler zu minimieren. Die Lösung im Frequenzbereich für den Wiener-Filter ist durch die folgende Übertragungsfunktion gegeben:

$H(u, v)$ ist die Degradationsfunktion, $H^*(u, v)$ ist ihre komplexe Konjugierte, $G(u, v)$ ist das verrauschte, unscharfe Bild, $N(u, v)$ ist das Leistungsspektrum des Rauschens und $I(u, v)$ ist das Leistungsspektrum des Originalbildes. $N(u, v) / I(u, v)$ ist bekannt als Rausch- zu Signalleistungsverhältnis. [15]

Der Wiener-Filter wird bei der Restaurierung von Realbildern nur selten eingesetzt, und zwar aus einem einfachen, aber entscheidenden Grund: Es handelt sich um ein lineares Verfahren. Bei einem

$$\hat{F}(u, v) = \left[\frac{H^*(u, v)}{|H(u, v)|^2 + \frac{N(u, v)}{I(u, v)}} \right] G(u, v)$$

linearen Verfahren sind die räumlichen Frequenzen, die in der Schätzung des echten Bildes enthalten sind, die des gemessenen Bildes, multipliziert mit einem bestimmten Faktor. Dies ist ausreichend, wenn alle räumlichen Frequenzen angemessen abgetastet werden. In den Fällen jedoch, in denen die räumlichen Frequenzen nicht gemessen werden, schätzt der Wiener-Filter sie einfach als 0. Lineare Verfahren können im Allgemeinen nicht extrapolieren oder interpolieren, um nicht gemessene räumliche Frequenzen zu schätzen. Techniken, die diese fehlenden Raumfrequenzen nicht schätzen, erzeugen Bilder mit deutlich sichtbaren Fehlern. Um eine gute Restaurierungstechnik zu entwickeln, müssen einige plausible Werte für diese fehlenden räumlichen Frequenzen geschätzt werden. Folglich sind nichtlineare Restaurierungsverfahren erforderlich. [54]

6.3.3 Methode der kleinsten Quadrate

Der eingeschränkte Kleinstquadrat-Filter (Constrained-Least-Squares-Filter (CLSF)) ist ein weiterer Ansatz, um einige der Schwierigkeiten des inversen Filters (übermäßige Verstärkung des Rauschens) und des Wiener-Filters (Schätzung des Leistungsspektrums des idealen Bildes) zu überwinden, wobei die Einfachheit eines räumlich unveränderlichen linearen Filters erhalten bleibt. [54]

Der CLSF ist eine Erweiterung des Wiener Filters, bei dem die Entfaltung keine Informationen über das Rauschen benötigt. Seine Lösung im Frequenzbereich ist durch die folgende Übertragungsfunktion gegeben:

$$\hat{F}(u, v) = \left[\frac{H^*(u, v)}{|H(u, v)|^2 + \gamma |P(u, v)|^2} \right] G(u, v)$$

γ ist ein Parameter, der manuell angepasst werden muss, um beste visuelle Ergebnisse zu erzielen, und $P(u, v)$ ist die Fourier-Transformierte des Laplacian-Operators $p(x, y)$:

$$p(x, y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Der Filter der eingeschränkten kleinsten Quadrate liefert im Vergleich zum Wiener Filter bessere Ergebnisse bei starkem und mittlerem Rauschen, und bei geringem Rauschen sind die Ergebnisse fast gleich. [15]

6.3.4 Gauß-Filter

Der Gauß-Filter ist ein besonderer Tiefpassfilter. Diese Art von Filter sorgt dafür, dass niedrige Frequenzen erhalten bleiben und schwächt hohe Frequenzen ab. Hohe Frequenzen kommen vor allem an Grenzen von Regionen vor, wo sich die Grauwerte stark unterscheiden. Daher wird der Gauß-Filter zum Glätten der Kanten genutzt, um dort die Unterschiede in den Grauwerten auszugleichen. Zusätzlich wird er zur Unterdrückung von Bildrauschen und als Weichzeichner eingesetzt.[4] [53]

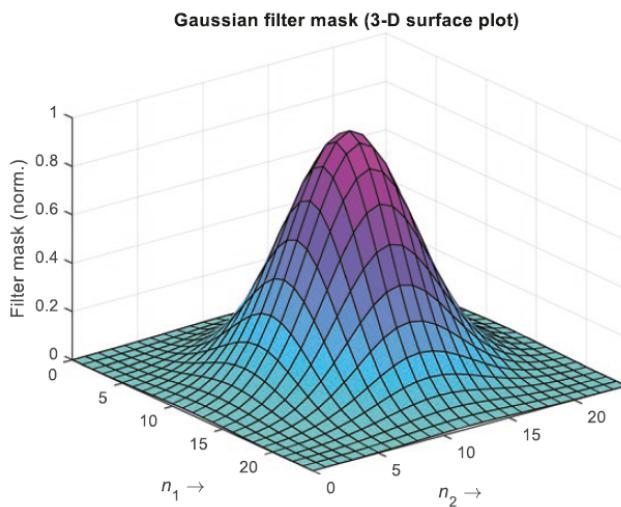


Abbildung 6.19: Abbildung des Gauß-Filters im dreidimensionalen Raum

Der Gauß-Filter basiert auf der bivariaten, rotationssymmetrischen Gaußschen Glockenkurve, wie in Abbildung 6.19 [53] gezeigt wird. Diese Kurve stellt die Filtermaske des Gauß-Filters dar. Da Bilder zweidimensional sind, basiert sie auf zwei Variablen x_1, x_2 und kann durch folgende Gleichung berechnet werden:

$$f(x_1, x_2) = \frac{1}{2\pi \cdot \sigma^2} \cdot \exp \left(-\frac{x_1^2 + x_2^2}{2 \cdot \sigma^2} \right)$$

Durch Verändern der Varianz σ kann die Stärke der Glättung reguliert werden. Nach dem Berechnen und Erstellen der Filtermaske kann diese zur Filterung an einem Bild angewandt werden. Abbildung 6.20 [55] zeigt die Filtermasken und die Anwendung des Gauß-Filters an einem Bild mit unterschiedlichen Varianzen. Je höher die Varianz, desto weicher werden die Grenzen. [53]

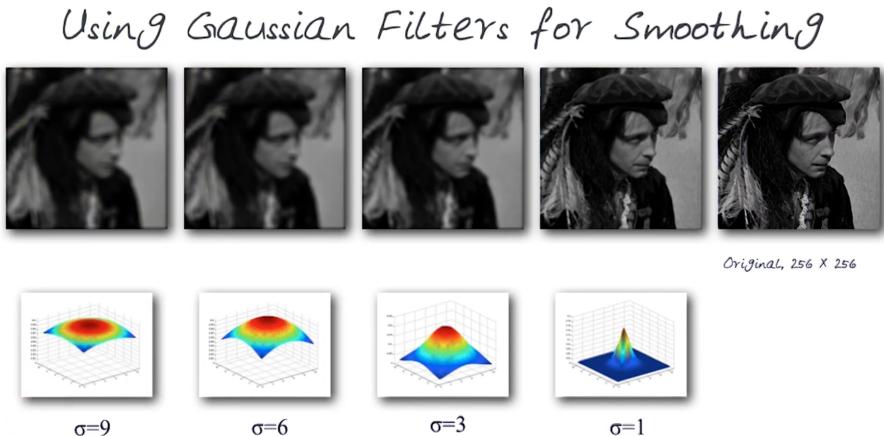


Abbildung 6.20: Anwendung des Gauß-Filters mit verschiedenen Varianzen

6.4 Zusammenfassung

In diesem Abschnitt werden die verschiedenen Algorithmen zur Merkmalerkennung und Restaurierung zusammengefasst. Die Segmentierungsalgorithmen können entweder Grenzen zwischen Regionen erkennen, zum Beispiel mithilfe des Canny Algorithmus und des Laplace Filters, oder die Region selbst wie beispielsweise bei der Wasserscheidentransformation, dem Schwellenwertverfahren oder dem Mean-Shift-Clustering. Die Letzteren eignen sich vermutlich besser, um fehlerhafte Bereiche eines Bildes zu erkennen.

Neben Segmentierungsalgorithmen wurden auch Algorithmen für die Texturanalyse untersucht, mit deren Hilfe sich Texturmuster erkennen lassen. Statistische Methoden nutzen die Verteilung der Grauwerte zur Beschreibung von Texturen. Beim strukturellen Ansatz werden Strukturelemente zur Darstellung komplexerer Strukturen verwendet. Der spektrale Ansatz beschreibt Gleichmäßigkeiten durch die spektrale Verteilung. Die untersuchten Algorithmen nutzen oft eine Kombination aus den oben genannten Methoden, z.B. wird bei der Energievariation eine Grauwertematrix erstellt, die zu den statistischen Methoden gehört, sowie das Local Binary Pattern angewandt, das zu dem strukturellen Ansatz gehört. Neben der Energievariation wurden das Binary Gabor Pattern, Morphologische Filter und Eigenfilter untersucht.

Zuletzt wurden Filter zur Bildrestaurierung betrachtet. Diese werden zur Rauschunterdrückung und Kantenglättung eingesetzt. Beschrieben wurde die Inverse Filterung, die Wiener Filterung, die Methode der kleinsten Quadrate und der Gauß-Filter.

7 Anwendung für die Bildreparatur (ImageRepair)

In diesem Kapitel werden die Anforderungen an die Anwendung, deren Ablauf sowie Details zur Implementierung beschrieben. Der Quellcode von der Anwendung ist im GitHub Repository unter <https://github.com/cschilling00/ImageRepair> zu finden.

7.1 Einführung und Anforderungen

Wie bereits im Abschnitt 2.2 geschildert, soll die Anwendung die geschädigten Bereiche von digitalen Bildern erkennen und durch das Zuschneiden entfernen. Die Statistik in Tabelle 2.1 zeigt, dass einfarbige Bilder und Bilder mit falsch angeordneten Abschnitten am häufigsten vorkommen. Aber anhand der durchgeföhrten Bibliotheken- und Algorithmenanalyse wurde zum aktuellen Zeitpunkt keine Lösung für die Reparatur der Bilder mit falsch angeordneten Abschnitten gefunden. Deshalb wird der Fokus der Anwendung zunächst auf die Bilder mit gestreiften und einfarbigen Bereichen gelegt. Daraus folgen funktionale und nichtfunktionale Anforderungen an die Anwendung.

7.1.1 Funktionale Anforderungen

In diesem Abschnitt werden die funktionalen Anforderungen an die Benutzeroberfläche sowie an die Verarbeitung der Bilder beschrieben.

7.1.1.1 Bildverarbeitung

- Nur Bilder im PNG oder JPEG/JPG Format können bearbeitet werden, da nur diese Formate erfahrungsgemäß auf den geschädigten Datenträgern vorkommen
- Ein/mehrere Bilder können empfangen werden
- Unterschiedliche Schäden können von dem Programm erkannt werden (hauptsächlich sollen Bereiche mit veränderter Farbe und gestreifte Bereiche erkannt werden)
- Der geschädigte Teil des Bildes soll abgeschnitten werden
- Falls kein Schaden am Bild erkannt wurde, soll es von dem Gesamtergebnis abgetrennt werden (in einen separaten Ordner gespeichert werden)

7.1.1.2 Benutzeroberfläche

- Es gibt eine Benutzeroberfläche, die die Bedienung der Applikation ermöglicht
- Es ist möglich, über die Pfadangabe ein/mehrere Bilder hochzuladen

- Alle Bilder können nach der Bearbeitung in einem angegebenen Ordner gespeichert werden
- Der Fortschritt der Bearbeitung wird angezeigt
- Fehlermeldungen werden angezeigt

7.1.2 Nichtfunktionale Anforderungen

- Einfache und intuitive Benutzeroberfläche, die nur notwendige Funktionalität abbildet
- Gute Geschwindigkeit der Bildverarbeitung (1-2 Sekunden pro Bild)
- Verständliche Fehlermeldungen
- Portabilität der Anwendung
- Wartbarkeit der Anwendung

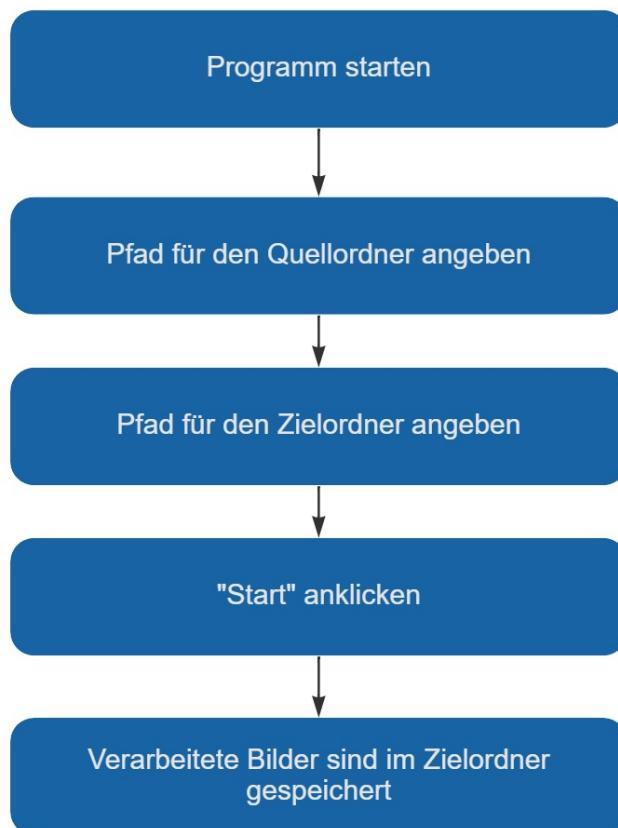
7.2 Ablauf der Bildverarbeitung

Abbildung 7.1 [eigene Quelle] zeigt einen für den Benutzer oder für die Benutzerin sichtbaren Programmablauf, der aus wenigen Schritten besteht und keine speziellen Kenntnisse erfordert. Ein weiteres Diagramm in der Abbildung 7.2 [eigene Quelle] demonstriert den Durchlauf des Hauptprogramms. Nachdem eine Anfrage von der Benutzeroberfläche erhalten wurde, wird der Prozess initiiert. Zunächst wird ein Zielordner für verarbeitete Bilder angelegt, falls noch keiner vorhanden ist. Bilder aus dem Quellordner werden nacheinander eingelesen. Wenn ein grauer (1) oder ein gestreifter (2) Bereich in einem Bild erkannt wird, wird er abgeschnitten. Es wird überprüft, ob (1) größer ist als (2) oder umgekehrt, damit entschieden werden kann, welches Ergebnis zu speichern ist. Es ist auch wichtig, dass der abzuschneidende Bereich tatsächlich ein geschädigter Bereich ist. Dafür ist ein Schwellenwert von 2.000.000 Pixel eingeführt worden. Bei Bildern im 4K Format stellt dieser Wert den Bereich dar, der beim Zuschneiden zusätzlich weggeschnitten wird. Dadurch wird sichergestellt, dass der geschädigte Bereich komplett entfernt wird.

Wenn der Bereich größer ist als der Schwellenwert, darf er abgeschnitten werden. Dadurch werden die Fälle ausgeschlossen, in denen ein unversehrtes Bild zugeschnitten wird. Die beschädigten Bereiche, die eine andere Farbe als grau haben, werden durch eine separate Methode erkannt, weil der Ablauf zusätzlich eine Kantenerkennung erfordert, was bei der grauen Farbe nicht notwendig ist.

7.3 Implementierung der Anwendung

Entsprechend dem Ergebnis der Bibliothekenanalyse im Kapitel 5 wird die Anwendung für die Bildreparatur (ImageRepair) in Python implementiert. Sie besteht aus zwei verknüpften Teilen: Benutzerober-



miro

Abbildung 7.1: Grober Programmablauf

berfläche (Graphical User Interface (GUI)) und Hauptprogramm. Zunächst wurde das Hauptprogramm (Main.py) entwickelt, im Anschluss wurde die Benutzeroberfläche (Gui.py) erstellt.

7.3.1 Benutzeroberfläche

Gemäß der Anforderung wurde die GUI schlicht und funktional gehalten und besteht aus einem Fenster, das in der Abbildung 7.3 [eigene Quelle] gezeigt ist.

Damit die Verarbeitung beginnt, sollen zwei Pfade angegeben werden: von dem Ordner mit den beschädigten Bildern und von dem, wo die verarbeiteten Bilder gespeichert werden sollen. Wenn der Quellordner nicht auffindbar ist, kommt eine Fehlermeldung (siehe Abbildung 7.4 [eigene Quelle]). Falls der Pfad des Zielordners nicht existiert, wird automatisch ein neuer Ordner angelegt. Rechts von dem Start Button befindet sich der Fortschrittsbalken.

Bei der Entwicklung der Benutzeroberfläche kamen folgende Python Module zum Einsatz:

- PyQt5 ist eine Bibliothek, mit der eine GUI entwickelt werden kann

- sys-Module stellt den Zugriff auf Variablen und Funktionen bereit, die vom Interpreter verwendet werden.
- time ist ein Modul, das zeitrelevante Funktionen beinhaltet.

7.3.2 Hauptprogramm

Die Python Bibliothek OpenCV stellt zahlreiche Algorithmen zur Verfügung. Für dieses Projekt sind diejenigen relevant, mit deren Hilfe beschädigte Bildbereiche erkannt werden können. Das heißt, zunächst wurden die Algorithmen für die Segmentierung und Texturenanalyse ausprobiert. Nach dem visuellen Vergleich der Ergebnisse wurde festgestellt, dass das optimalste Ergebnis zur Kantenerkennung vom Canny Algorithmus geliefert wird. Dadurch wurde die Behauptung aus Abschnitt 6.2.1.4 bestätigt, dass dieser Algorithmus die Bildkanten besser lokalisieren kann als die anderen. Außerdem kam das Thresholding Verfahren oft zum Einsatz (s. Abschnitt 6.2.1.3), um relevante Regionen des Bildes abzugrenzen.

Die entwickelte Anwendung ist in der Lage, graue, bunt gestreifte und einfarbige Bereiche zu erkennen. Nachdem ein solcher Bereich detektiert wurde, wird er abgeschnitten, und das Bild wird in den Zielordner gespeichert. Falls kein Schaden festgestellt wurde, wird das Bild in einem separaten Ordner abgelegt.

Das Hauptprogramm wurde mithilfe von folgenden Modulen entwickelt:

- cv2 stellt die Funktionalität von OpenCV bereit
- numpy beinhaltet mathematische Funktionen, ermöglicht die Arbeit mit Arrays
- os ist ein Modul für die Kommunikation mit dem Betriebssystem (plattformübergreifend)
- glob-Modul arbeitet mit Dateipfaden

7.4 Zusammenfassung

In diesem Kapitel wurden Anforderungen an die Benutzeroberfläche und an die Bildverarbeitung gestellt. Im Anschluss wurde der Ablauf der Anwendung für Nutzer sowie der Ablauf des Hauptprogramms in Diagrammen dargestellt. Danach wurde die Implementierung der Anwendung und die Nutzung von weiteren Bibliotheken beschrieben. Im folgenden Abschnitt werden die Anwendung und die Ergebnisse der bearbeiteten Bilder bewertet und die Arbeit wird reflektiert.

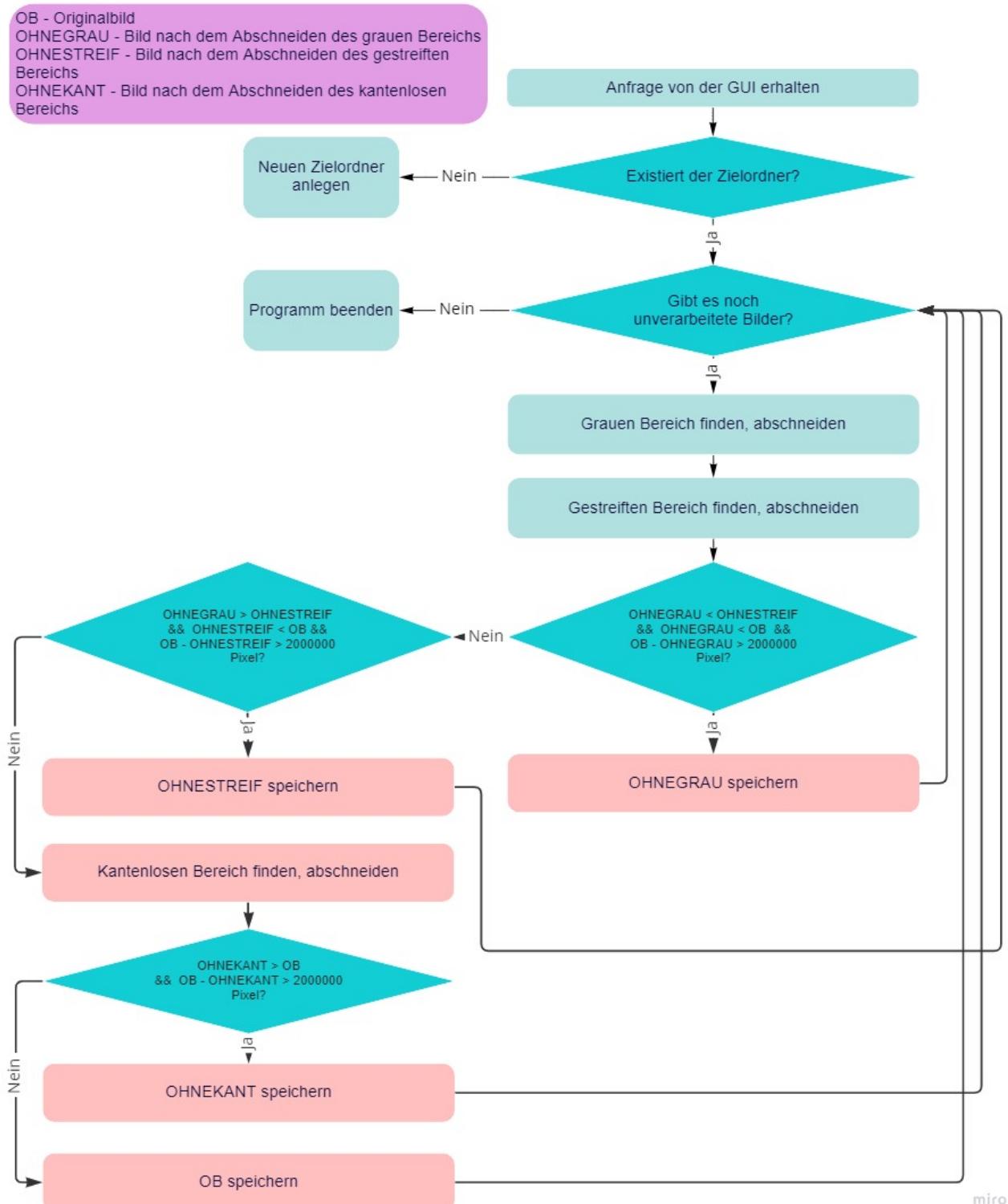


Abbildung 7.2: Hauptprogrammablauf

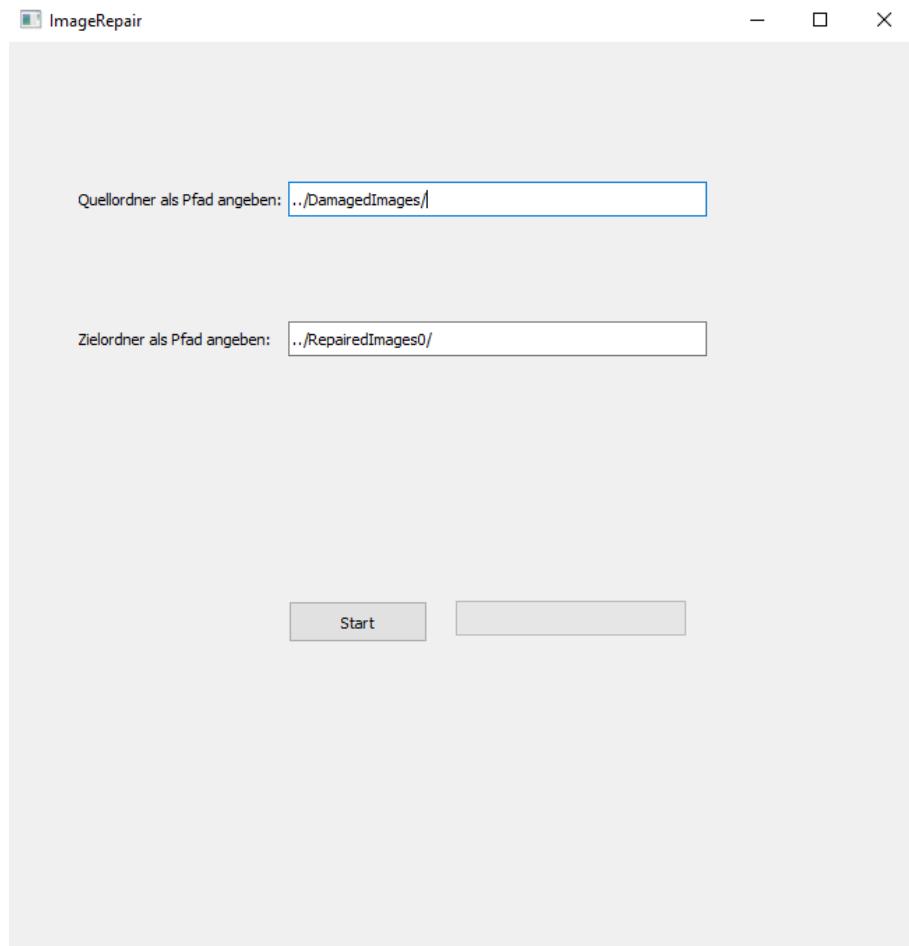


Abbildung 7.3: Benutzeroberfläche von ImageRepair

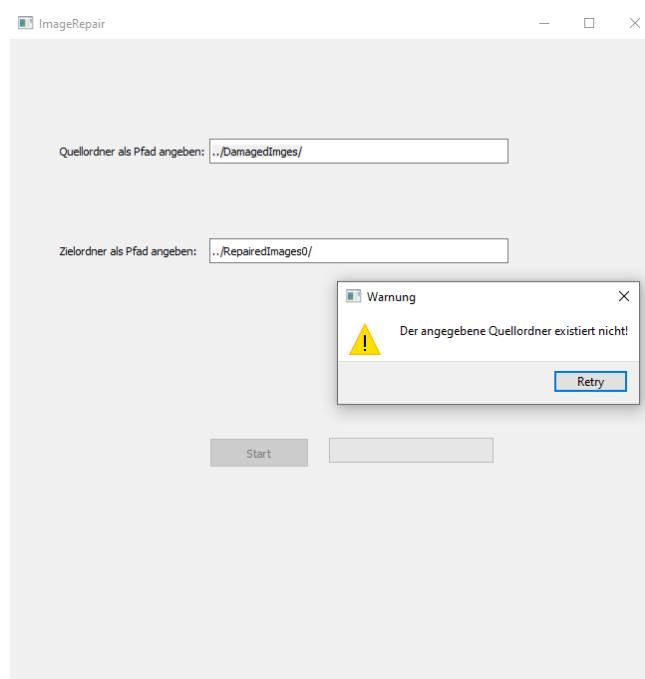


Abbildung 7.4: Fehlermeldung

8 Evaluierung der Ergebnisse und Reflexion

In diesem Kapitel wird die Anwendung und die Bearbeitung der Bilder bewertet. Danach wird die Arbeit reflektiert.

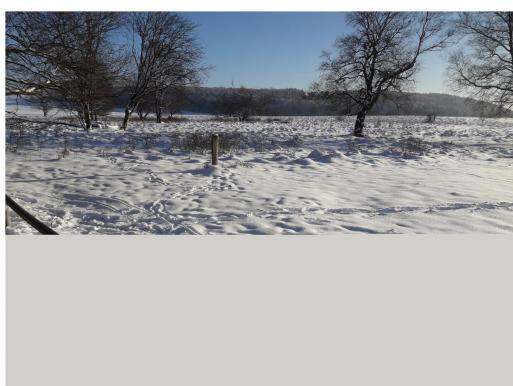
8.1 Testergebnisse und Bewertung

Um herauszufinden, wie schnell ein Bild im Durchschnitt verarbeitet wird, wurde ImageRepair mit vier unterschiedlich großen Bildermengen getestet. Folgende Ergebnisse wurden auf einem Rechner mit dem Prozessor Intel Core i7-10510U und 8 GB Random-Access-Memory (RAM) erzielt:

- 10 Bilder 890 MB in 12,61 Sekunden (1 Bild in 1,26 Sekunden)
- 20 Bilder 956 MB in 20,35 Sekunden (1 Bild in 1,02 Sekunden)
- 50 Bilder 1,11 GB in 63,95 Sekunden (1 Bild in 1,28 Sekunden)
- 100 Bilder 3,51 GB in 109,97 Sekunden (1 Bild in 1,11 Sekunden)

Dadurch ergibt sich eine durchschnittliche Dauer der Verarbeitung eines Bilds von 1,16 Sekunden. Sie entspricht der Anforderung bezüglich der Geschwindigkeit aus dem Abschnitt 7.1.2.

Abbildungen (alle drei kommen aus eigener Quelle) 8.1 , 8.2 und 8.3 zeigen Bilder mit drei von den vier Schadenstypen, die im Abschnitt 2.1 erwähnt wurden. Bilder mit falsch angeordneten Abschnitten können von der Anwendung noch nicht erkannt werden. Falls sie in der zu verarbeitenden Bildermenge auftauchen, werden sie von ImageRepair als ungeschädigt betrachtet und zusammen mit anderen unveränderten Bildern in einem getrennten Ordner gespeichert.



(a) Originalbild



(b) Bild nach Nutzung der Anwendung

Abbildung 8.1: Bild mit Graubereich

Die Schäden wurden erkannt und aus den Bildern entfernt. Ob die Ergebnisse weiter verwendet werden, wird vom Kunden oder von der Kundin entschieden. Die Bilder werden lediglich mit den restlichen geretteten Daten abgeschickt.

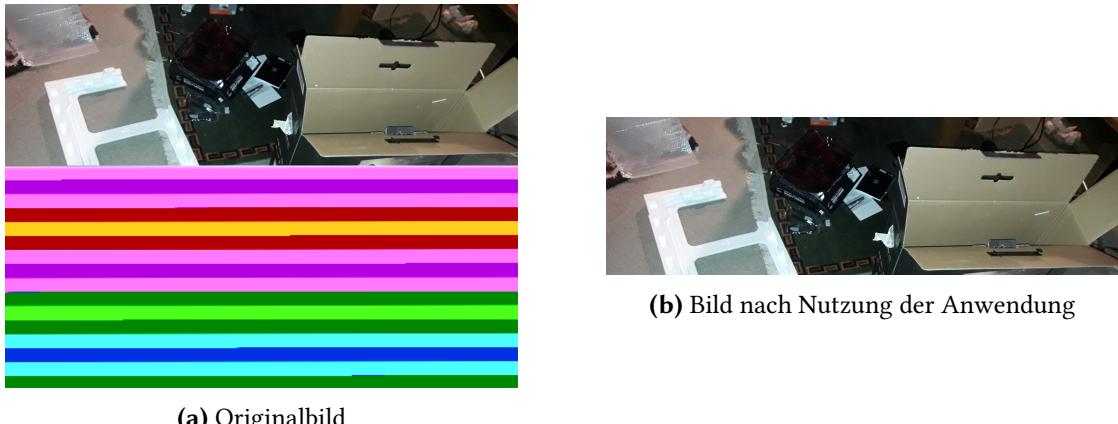


Abbildung 8.2: Bild mit gestreiftem Bereich

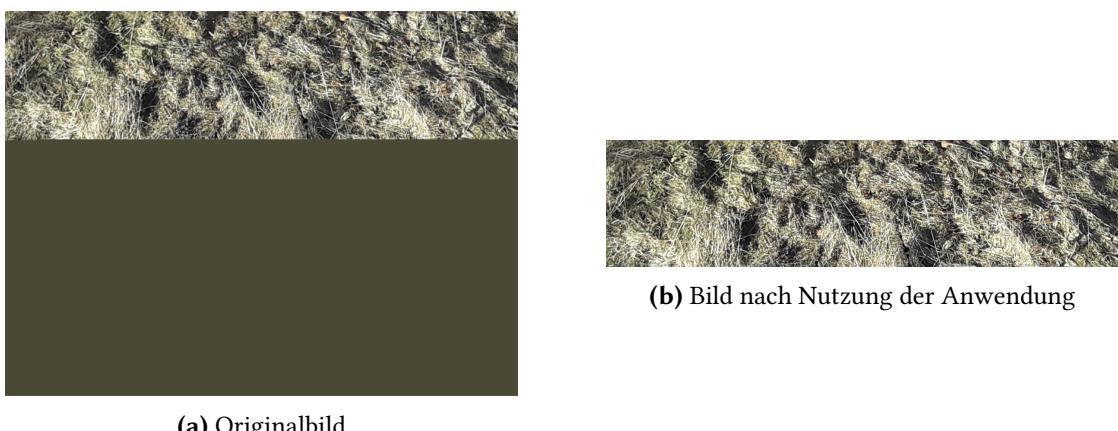


Abbildung 8.3: Bild mit einfarbigem Bereich

8.2 Reflexion

In den Abschnitten 1.2 und 2.2 wurden Ziele für diese Arbeit gesetzt, und zwar die Verbesserung des Kenntnisstandes durch die Untersuchung der Technologien und Algorithmen für die Bildverarbeitung und die Entwicklung einer Anwendung zur Bildreparatur. Es wurde entdeckt, wie groß das Spektrum von Algorithmen ist, die von verschiedenen Bibliotheken bereitgestellt werden: von der Anwendung eines Filters auf ein Bild bis hin zu Videoverarbeitung in der Echtzeit. Die gesammelten Kenntnisse konnten in dem darauffolgenden Entwicklungsprozess eingesetzt werden. Es ließ sich feststellen, dass unterschiedliche Algorithmen, die ähnliche Funktion haben, unterschiedlich gute Ergebnisse liefern. Es wurde gezeigt, welche Bildschäden bei der Datenrettung am meisten vorkommen (s. Abschnitt 2.1). Obwohl es nicht gelungen ist, alle Arten der Schäden zu beseitigen, wurde durch die entwickelte Anwendung ermöglicht, die Leistung der Datenrettung zu verbessern. Der Kunde oder die Kundin bekommt dadurch mehr unbeschädigte Bilder.

9 Zusammenfassung und Ausblick

Die vorliegende Arbeit beschäftigte sich mit der Bearbeitung korrupter Bilder im JPEG und PNG Format. Dazu wurde eine Anwendung entwickelt, die eine größere Menge an Bildern einliest, analysiert und die als korrupt erkannten Bereiche entfernt. Es existiert bereits eine Anwendung namens JPEG Medic [56], die Bilder im JPEG Format auf Basis von MCU Blöcken bearbeiten lässt, dies muss allerdings manuell und kostenpflichtig vom Betreiber durchgeführt werden. Die in dieser Arbeit entwickelte Anwendung steht opensource zur Verfügung, bietet aber keine Funktionen zum reparieren, sondern nur zum Entfernen dieser Bereiche. Der Entwicklung der Anwendung ging eine ausführliche Untersuchung verschiedener Bibliotheken zur Bildverarbeitung sowie einer geeigneten Programmiersprache voraus. Die Bibliothek OpenCV verfügt über ein großes Set an Features und ist in den Sprachen Java, C++ und Python anwendbar. Als Sprache wurde Python ausgewählt, da sie laut der Stackoverflow Survey 2021 am beliebtesten ist und Vorteile der weiteren für Python erstellten Bibliotheken für die Bildverarbeitung mit sich bringt. Im Anschluss wurden verschiedene Algorithmen zur Segmentierung und Textanalyse näher beschrieben, auf deren Grundlage die Anwendung entwickelt wurde. Die relevantesten Algorithmen für die Anwendung waren der Canny Algortihmus zur Kantenerkennung sowie das Thresholding Verfahren. Desweiteren wurde die Maskierung, Konturenerkennung sowie Segmentierung anhand von Konturenerkennung eingesetzt. Die Anwendung besteht aus einer graphischen Benutzeroberfläche und dem Hauptprogramm, das im Hintergrund die Bildverarbeitung durchführt. Der Nutzer kann die Pfade zu dem Quellordner und Zielordner angeben. Drückt er auf den Startknopf, beginnt das Hauptprogramm mit der Verarbeitung. Der Fortschritt wird den Nutzern anhand eines Fortschrittsbalken angezeigt. Die Anwendung erkennt 3 verschiedene Arten von Schäden: Graubereiche, Streifen und weitere einfarbige Bereiche. Die Verarbeitungsgeschwindigkeit auf einem Rechner mit dem Prozessor Intel Core i7-10510U und 8 GB RAM liegt durchschnittlich bei 1,16 Sekunden pro Bild.

Obwohl die meisten Ziele der Arbeit erreicht wurden, bleibt die ImageRepair Anwendung ausbaufähig. Moderne Technologien für Bild- und Videoverarbeitung erlauben unterschiedlichste Manipulationen. Man sollte das nutzen, um neue Bildformate von ImageRepair bearbeiten zu lassen und den Prozess durch Parallelisierung zu beschleunigen, falls die von geschädigten Datenträgern kommende Bildermenge steigt. Auch können Features wie die der Anwendung JPEG Medic mit eingebaut werden, die es dem Nutzer oder der Nutzerin erlauben, die Bilder zuerst versuchen zu reparieren. Es wurde festgestellt, dass beim Zuschneiden der Bilder Metadaten verloren gehen. Die Applikation könnte so angepasst werden, dass diese erhalten bleiben, denn manchen Kunden ist das wichtig. Bei den Arten von Schäden, die ImageRepair beseitigen kann, gibt es auch ein Erweiterungspotential. Falsch angeordnete Abschnitte eines Bilds können aktuell programmatisch nicht erkannt und repariert

werden. Vermutlich könnte es mithilfe der Möglichkeiten der künstlichen Intelligenz erreicht werden. Hierzu wird eine zusätzliche Einarbeitung gebraucht.

Literaturverzeichnis

- [1] Marcamillion, "Why do images get "corrupted"?" *Photography*, 2010 [Online]. Adresse: <https://photo.stackexchange.com/questions/6045/why-do-images-get-corrupted>.
- [2] S. Dey, in *Hands-On Image Processing with Python*. Birmingham, UK: Packt Publishing, 2018.
- [3] B. Rohrer, "How to Convert an RGB Image to Grayscale," *Signal Processing Techniques*, 2019, Nov [Online]. Adresse: https://e2eml.school/convert_rgb_to_grayscale.html.
- [4] Y.-J. Zhang, "Color Image Processing, Image Pattern Recognition, Segmentation Introduction, Pixel Spatial Relationship, Object Segmentation Methods, Edge Detection, Texture Analysis, Mathematical Morphology for Binary Images, Mathematical Morphology for Binary Images, Image Pattern Recognition, Image Restoration, Filtering Techniques," in *Handbook of Image Engineering*. Singapore, Singapore: Springer Nature Singapore Pte Ltd., 2021.
- [5] H.-G. Schiele, "Farben," in *Computergrafik für Ingenieure: Eine anwendungsorientierte Einführung*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.
- [6] Wikipedia, "HSL and HSV," 2022, [Online]. Adresse: https://en.wikipedia.org/wiki/HSL_and_HSV (besucht am 09.03.2022).
- [7] "Farbart," *ITWissen*, 2012, [Online]. Adresse: <https://www.itwissen.info/Farbart-chroma.html>.
- [8] Isaac, "Corrupted JPG has undesired color bars overlay," *Photography*, 2016 [Online]. Adresse: <https://photo.stackexchange.com/questions/73520/corrupted-jpg-has-undesired-color-bars-overlay?noredirect=1&lq=1>.
- [9] J. van Steen, "Corrupt JPEG Files and how to fix them," *DiskTuna*, 2021, Dec [Online]. Adresse: <https://www.disktuna.com/corrupt-jpeg-files/>.
- [10] A. Distante und C. Distante, "Canny algorithm, Gradient Filter, Laplacian Operator," in *Handbook of Image Processing and Computer Vision. Volume 2: From Image to Pattern*. Cham, Switzerland: Springer Nature Switzerland AG, 2020.
- [11] N. Otsu, "A threshold selection method from gray level histograms," in *IEEE Transactions on Systems, Man, and Cybernetics* 9, 1979, S. 62–66.
- [12] L. Zhang, Z. Zhou und H. Li, "Binary Gabor pattern: An efficient and robust descriptor for texture classification," in *2012 19th IEEE International Conference on Image Processing*, 2012, S. 81–84.
- [13] S. Fekri Ershad, "Texture Classification Approach Based on Energy Variation," *international journal of multimedia technology*, Jg. 2, S. 52–55, Jan. 2012.
- [14] A. Tkacenko, P. Vaidyanathan und T. Nguyen, "On the eigenfilter design method and its applications: a tutorial," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Nr. 9, S. 497–517, 2003.
- [15] S. Shruthi und K. Satheeshkumar, "Constrained least squares filtering followed by denoising of decomposed images using wave atom and wavelet transform," *Procedia Computer Science*, Jg. 115, S. 526–532, 2017.
- [16] S. Ulili, "How To Process Images in Node.js With Sharp," *DigitalOcean*, 2021, Sept [Online]. Adresse: <https://www.digitalocean.com/community/tutorials/how-to-process-images-in-nodejs-with-sharp>.

- [17] L. Fuller, “sharp,” 2022 [Online]. Adresse: <https://sharp.pixelplumbing.com/> (besucht am 09.03.2022).
- [18] J. Cupitt, “libvips : an image processing library,” 2022 Feb. 22 [Online]. Adresse: <https://github.com/libvips/libvips/blob/master/README.md> (besucht am 09.03.2022).
- [19] Qix, “color,” 2022 Mar [Online]. Adresse: <https://www.npmjs.com/package/color> (besucht am 09.03.2022).
- [20] R. LeFevre, “What is CamanJS?,” 2013, Jul. 27 [Online]. Adresse: <http://camanjs.com/> (besucht am 09.03.2022).
- [21] ——, “Basic Usage,” 2013, Jul. 27 [Online]. Adresse: <http://camanjs.com/guides/#BasicUsage> (besucht am 09.03.2022).
- [22] M. Shokeen, “Creating an Image Editor Using CamanJS: Applying Basic Filters,” *envatotuts+*, 2018, Mar [Online]. Adresse: <https://code.tutsplus.com/tutorials/creating-an-image-editor-using-camanjs-applying-basic-filters--cms-30251>.
- [23] E. Gouillart, “Scikit-image: image processing,” 2020, Sept. [Online]. Adresse: <https://scipy-lectures.org/packages/scikit-image/index.html#id14> (besucht am 09.03.2022).
- [24] Tutorials und Examples, “Python Scikit-image | Image Processing Using Scikit-Image,” 2021, Jul. 03 [Online]. Adresse: <https://www.tutorialandexample.com/python-scikit-image-image-processing-using-scikit-image> (besucht am 09.03.2022).
- [25] scikit-image development team, “Image data types and what they mean,” 2022, [Online]. Adresse: https://scikit-image.org/docs/stable/user_guide/data_types.html (besucht am 09.03.2022).
- [26] ——, “A crash course on NumPy for images,” 2022, [Online]. Adresse: https://scikit-image.org/docs/stable/user_guide/numpy_images.html (besucht am 09.03.2022).
- [27] ——, “Geometrical transformations of images,” 2022, [Online]. Adresse: https://scikit-image.org/docs/stable/user_guide/geometrical_transform.html#cropping-resizing-and-rescaling-images (besucht am 09.03.2022).
- [28] ——, “Image Segmentation,” 2022, [Online]. Adresse: https://scikit-image.org/docs/stable/user_guide/tutorial_segmentation.html (besucht am 09.03.2022).
- [29] ——, “Module: feature,” 2022, [Online]. Adresse: <https://scikit-image.org/docs/dev/api/skimage.feature.html> (besucht am 09.03.2022).
- [30] M. Roynard, E. Carlinet und T. Géraud, “A Modern C++ Point of View of Programming in Image Processing,” 2022, Feb [Online]. Adresse: <https://hal.archives-ouvertes.fr/hal-03564252/document> (besucht am 09.03.2022).
- [31] G. Bradski und A. Kaehler, “Overview, Getting to know OpenCV,” in *Learning OpenCV*. Sebastopol, USA: O’Reilly Media, 2008.
- [32] Doxygen, “Feature Detection and Description,” 2022, [Online]. Adresse: https://docs.opencv.org/4.x/db/d27/tutorial_py_table_of_contents_feature2d.html (besucht am 09.03.2022).
- [33] openbase, “Search results for ‘image proceccing’ for JavaScript,” 2022, [Online]. Adresse: <https://openbase.com/search?q=image%5C%20proceccing> (besucht am 09.05.2022).
- [34] stackoverflow, “What is the best JavaScript image processing library? [closed],” 2010, [Online]. Adresse: <https://stackoverflow.com/questions/3351122/what-is-the-best-javascript-image-processing-library> (besucht am 09.05.2022).

- [35] ——, “Developer Survey 2021,” 2021, [Online]. Adresse: <https://insights.stackoverflow.com/survey/2021#technology> (besucht am 09. 05. 2022).
- [36] Z. Akmal, “how to retrieve images from sub folders and store them in another separate folder?,” 2019, Apr [Online]. Adresse: <https://stackoverflow.com/questions/55552537/how-to-retrieve-images-from-sub-folders-and-store-them-in-another-separate-folde> (besucht am 09. 05. 2022).
- [37] Quora, “How can I read multiple images in Python presented in a folder?,” 2018, [Online]. Adresse: <https://www.quora.com/How-can-I-read-multiple-images-in-Python-presented-in-a-folder> (besucht am 09. 05. 2022).
- [38] S. Träger, “Erkennung von Fahrbahnmarkierungen – Teil 2: Bildsegmentierung und wie sie genutzt werden kann,” 2021, Feb. 08 [Online]. Adresse: <https://blog.doubleslash.de/erkennung-von-fahrbahnmarkierungen-teil-2-bildsegmentierung-und-wie-sie-genutzt-werden-kann/> (besucht am 09. 03. 2022).
- [39] R. Pichumani, “Boundary-Based Segmentation,” 1997, Jul. 07 [Online]. Adresse: https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RAMANI1/node24.html (besucht am 09. 03. 2022).
- [40] K. Rajamani, B. C C und L. V L, “Brain Tumor Extraction from MRI Brain Images Using Marker Based Watershed Algorithm,” Aug. 2015.
- [41] scikit-image development team, “Watershed segmentation,” 2022, [Online]. Adresse: https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_watershed.html (besucht am 09. 03. 2022).
- [42] M. Nedrich, “Mean Shift Clustering,” 2015, May [Online]. Adresse: <https://spin.atomicobject.com/2015/05/26/mean-shift-clustering/> (besucht am 09. 03. 2022).
- [43] Y. Wu, “Mean shift pixel cluster,” 2022, [Online]. Adresse: <https://de.mathworks.com/matlabcentral/fileexchange/40990-mean-shift-pixel-cluster> (besucht am 09. 03. 2022).
- [44] Wikipedia, “Schwellenwertverfahren,” 2022, [Online]. Adresse: <https://de.wikipedia.org/wiki/Schwellenwertverfahren> (besucht am 09. 03. 2022).
- [45] T. Birdal, “Famous Otsu Thresholding in C,” 2009, Jul [Online]. Adresse: <https://www.codeproject.com/Articles/38319/Famous-Otsu-Thresholding-in-C> (besucht am 09. 03. 2022).
- [46] Wikipedia, “Canny edge detector,” 2022, [Online]. Adresse: https://en.wikipedia.org/wiki/Canny_edge_detector (besucht am 09. 03. 2022).
- [47] S. Perkins und P. Marais, “Identification and reconstruction of bullets from multiple X-rays,” Jan. 2006.
- [48] T. Ojala, M. Pietikäinen und D. Harwood, “A comparative study of texture measures with classification based on featured distributions,” *Pattern Recognition*, Jg. 29, Nr. 1, S. 51–59, 1996. Adresse: <https://www.sciencedirect.com/science/article/pii/0031320395000674>.
- [49] Wikipedia, “Opening (Bildverarbeitung),” 2022, [Online]. Adresse: [https://de.wikipedia.org/wiki/Opening_\(Bildverarbeitung\)](https://de.wikipedia.org/wiki/Opening_(Bildverarbeitung)) (besucht am 09. 03. 2022).
- [50] ——, “Closing,” 2022, [Online]. Adresse: <https://de.wikipedia.org/wiki/Closing> (besucht am 09. 03. 2022).
- [51] J. X.Jane und S. P. J., “Fundaments for free-form surfaces,” in *Advanced Metrology: Freeform Surfaces*, London, UK: Academic Press, 2020.

- [52] H. Tim, “Gray Level Co-occurrence Matrix,” 2022, [Online]. Adresse: <https://github.com/JuliaImages/ImageFeatures.jl/blob/master/docs/src/tutorials/glcm.md> (besucht am 09.03.2022).
- [53] M. Werner, “Filtern und Entzerren mit der 2-D-DFT, Gauß-Filter,” in *Digitale Bildverarbeitung*. Wiesbaden, Germany: Springer Fachmedien Wiesbaden GmbH, 2021.
- [54] R. Lagendijk und J. Biemond, “Basic Methods for Image Restoration and Identification,” in *Handbook of Image and Video Processing (Second Edition)*. Burlington, USA: Elsevier Academic Press, 2005.
- [55] Udacity, *Using Gaussian Filters for Smoothing Cont*, 2015, Feb [Online Video]. Adresse: <https://www.youtube.com/watch?v=4RpDOAbnNYE> (besucht am 09.03.2022).
- [56] D. Anisimov, *JpegMedic - Repair Damaged JPEG Images Like a Pro*. Adresse: <https://www.jpegmedic.com/tools/jpegmedic/> (besucht am 24.05.2022).