UNIVERSITÄT
KOBLENZ · LANDAU

Erweiterung der Music-Recognition-Software AudiVeris durch Bildvorverarbeitung

Bachelorarbeit

zur Erlangung des Grades Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Ruwen Davis Bergen

Erstgutachter:	Prof. Dr.-Ing. Dietrich Paulus (Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau)
Zweitgutachter:	Detlev Droege (Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau)

Kurzfassung

Ziel dieser Bachelorarbeit war es, in die Musiknoten-Erkennungs-Software AudiVeris eine Bildvorverarbeitung einzubauen, damit auch aus fehlerbehafteten Notenbildern Daten extrahiert werden können.

Der Ablauf startet mit einer Binarisierung durch ein regionales Otsu-Verfahren. Daraufhin wird das Notenblatt nach etwaigen Krümmungen abgesucht, wie sie z.B. eine Buchfalz verursachen würde. Dazu wird die Hough-Transformation zur Linienfindung und der K-Means-Algorithmus zur Cluster-Detektion verwendet. Aufbauend wird das Notenbild unter Benutzung der gefundenen Krümmung geebnet.

Abstract

The goal of this bachelor thesis was to add an image processing step to the music recognition software AudiVeris, in order to extract data even from faulty music sheet images.

The procedure starts with a binarization using a regional version of Otsu's method. Following this the music sheet is searched for possible bendings, similar to those a hardcover book would cause. To achieve this the Hough transform is used for line detection and the k-means algorithm for cluster detection. Thereafter the music image is straightened using the discovered curvature.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Vereinbarung der Arbeitsgruppe für Studien- und Abschlussarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	ja ☐	nein ☒
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	ja ☒	nein ☐

Koblenz, den 06. Mai 2019

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Im Alltag von Musikern, Sängern und Musikbegeisterten kommt es vor, dass Partituren, Lieder und Stücke angepasst werden müssen. Es entsteht der Bedarf Passagen an den eigenen Gebrauch anzupassen, die Tonlage zu verändern oder Melodien zu vereinfachen oder auszuweiten. Oft liegen dabei die Noten nicht in digitaler sondern in ausgedruckter oder rasterbildbasierter Form vor.

Um diesem Bedarf entgegen zu kommen gibt es Musik-Erkennungs-Software wie beispielsweise AudiVeris. Doch nicht immer gelingt es der Software vorhandene Noten zu erkennen und die Daten richtig zu extrahieren.

Diese Arbeit beschäftigt sich damit, wie verunreinigte Notenblätter nutzbar gemacht und wie die Möglichkeiten, Noten zu erkennen, erweitert werden können. Hierzu wird eine andere Binarisierungsmethode verwendet, als sie derzeit in AudiVeris implementiert ist. Außerdem wird über einen Linienfindungsalgorithmus untersucht, ob das Bild eine Krümmung hat. Darauffolgend soll es begradigt werden.

In dem folgenden Kapitel 2 wird die Projektstruktur der Software AudiVeris erklärt. Außerdem werden die Algorithmen, die in dieser Arbeit genutzt werden erläutert. Anschließend wird dargestellt, wie diese Algorithmen eingebaut wurden und welche Methoden zur Unterstützung der Funktionen benutzt wurden.

Im Kapitel 4 werden Tests durchgeführt, um zu ermitteln, ob und wie weit sich die Notenerkennung verbessert hat. Hierzu wird ein Testnotenblatt auf mehrere Arten synthetisch verunreinigt und das Erkennungsprogramm darauf angewendet.

2 | Theoretischer Teil

2.1 Einleitung

In diesem Kapitel werden die Anforderungen an die Bildvorverarbeitung im Programm AudiVeris festgelegt. Die Projektstruktur der Software wird vorgestellt und relevante Algorithmen, wie die *Otsu-Segmentierung* und die *Hough-Transformation* werden erklärt.

2.1.1 Anforderungen

Notenblätter stehen oft in niedriger Auflösung oder als verrauschte Scans zur Verfügung. Die innere Falte eines gebundenen Buches lässt das Musikstück nicht eben auf dem Scanner aufliegen und das Bild ist verzerrt. Diese Arbeit soll die Notenblätter vorverarbeiten, sodass die weiteren Schritte in AudiVeris die Noten extrahieren können. Insbesondere soll die Notenerkennung bei

- Ungleicher Belichtung
- Rauschen
- Krümmungen im Bildmaterial

funktionieren und Ergebnisse liefern.

2.2 AudiVeris

AudiVeris ist eine Open-Source-Software zur Musikerkennung. Das Projekt wurde gestartet und ist unter der Leitung von Hervé Bitteur.

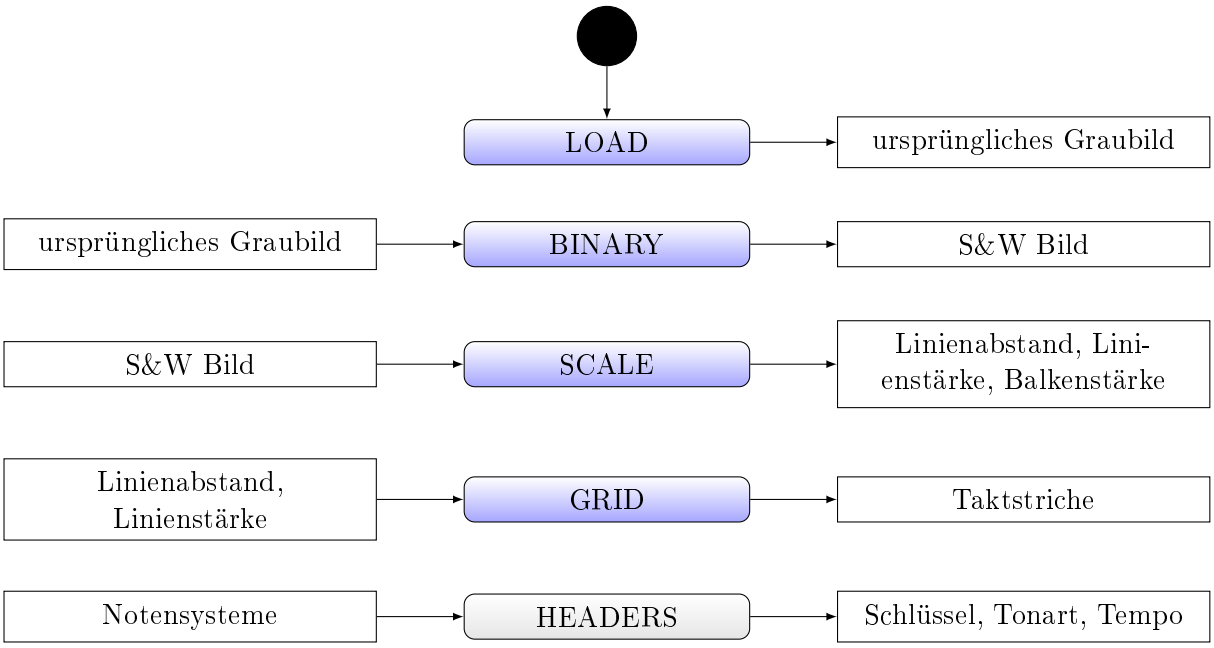
2.2.1 Projektstruktur

In AudiVeris sind die Dateien in einem *Book* organisiert, welches mehrere Seiten (*Sheets*) haben kann. Der Benutzer kann die Notenerkennung auf eine Seite oder auf ein ganzes Buch anwenden. Das laufende Projekt kann auch als Projektdatei gespeichert werden. Die gespeicherten Daten sind das Ergebnis des XML-Marshalling des Buches und seiner Seiten (engl. *to marshal* bedeutet ordnen)([Bit17]).

Die Projektdateien werden unter dem Format .omr gespeichert. Dies ist ein Zip-Format und kann einfach dekomprimiert, inspiziert und manuell verändert werden. Es enthält eine Datei *Book.xml* die die übergeordnete Struktur des Buches organisiert. Außerdem sind das lauflängenkodierte Binärbild, eine Datei für die Positionen aller Notenköpfe und eine Datei mit allen anderen gefundenen Informationen Teil der .omr-Datei.

Wenn mehrere Projekte in AudiVeris geöffnet sind, behält das Programm nur die Daten der Bücher-Dateien im Arbeitsspeicher. Die Informationen der Seiten werden geladen, wenn das übergeordnete Buch ausgewählt wird.

Abbildung 2.1: Pipeline in AudiVeris (Auszug 1)

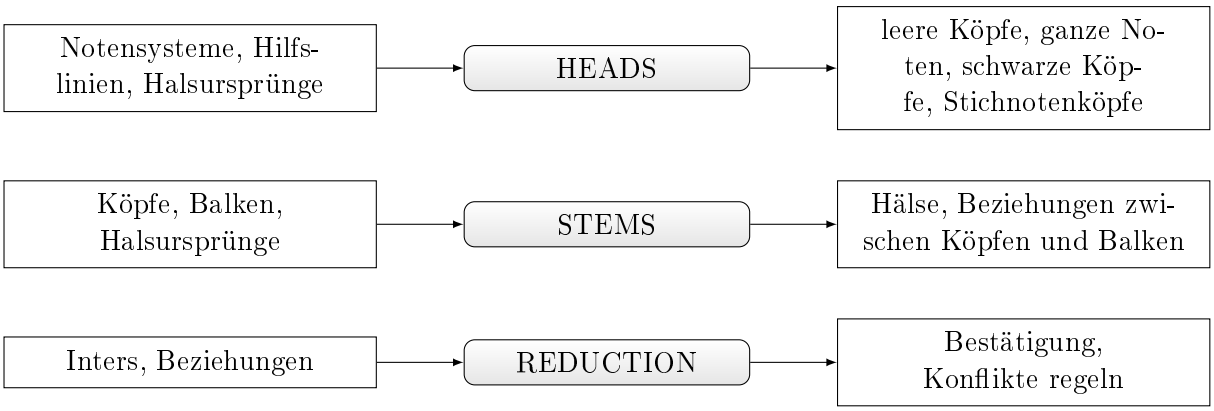


2.2.2 Pipeline in AudiVeris

Die Reihenfolge der Operationen (engl. *pipeline*) ist im Diagramm 2.1 auf dieser Seite beschrieben:

Die in der Mitte dargestellten Schritte sind im AudiVeris-Programmablauf als *Steps* bezeichnet (auf der nächsten Seite). Dabei führen die blau hinterlegten Schritte Rechnungen auf Bildern aus, während die grauen mit verschiedenen bereits extrahierten Daten arbeiten. Der *Load-Schritt* lädt ein Bild aus der Datei. Hierbei werden Bilder mit Farbkanälen direkt in Grauwertbilder umgewandelt. Das Bild wird nicht im Arbeitsspeicher behalten, stattdessen speichert sich die Klasse *Picture* nur den Dateipfad und lädt das Bild bei Anfrage neu. Der *Binary-Schritt* ist die einzige Bildvorverarbeitung die in AudiVeris verbaut ist: sie binarisiert das ursprüngliche Grauwertbild.

Abbildung 2.2: Pipeline in AudiVeris (Auszug 2)



Die beiden Schritte *Scale* und *Grid* verändern das Binärbild nicht weiter sondern suchen Informationen, die für die Folgeschritte notwendig sind.

Der Großteil der grau hinterlegten Schritte dient der Merkmalsextraktion. Eine besondere Stellung nimmt dabei der Schritt *Reduction* ein. In den vorangehenden Schritten werden alle gefundenen Merkmale mit ihrer Übereinstimmungswahrscheinlichkeit abgespeichert. In dem Reduktionsschritt wird die Entscheidung getroffen, welche Symbole (Glyphen, s. 2.2.3) verworfen und welche als wahr (Inter, s. 2.2.3) angenommen werden.

2.2.3 Einige Klassen in AudiVeris

In diesem Abschnitt werden einige Klassen und Strukturen in AudiVeris beleuchtet, die für die Arbeit von Wichtigkeit sind.

Die Klassengruppe „Step“

Alle in Abbildungen 2.1 und 2.2 abgebildeten Schritte sind Erben der Klasse *AbstractStep*. Sie haben alle die Methode `doit()`, in welcher der Prozess abläuft, für den der Schritt verantwortlich ist. Die Methode `displayUI()` ist ebenfalls in der Elternklasse vorhanden, wird aber nicht von jedem Erben genutzt. Hier kann das Ergebnis auf der Benutzeroberfläche dargestellt werden. AudiVeris versucht nur das Nötigste darzustellen, vermutlich um möglichst wenig Arbeitsspeicher zu gebrauchen. Die dargestellten Bilder sind das Ursprungsbild, das Binärbild und die gefundenen Daten am Ende aller Schritte.

Die Klasse „Picture“

Die Klasse *Picture* hält den Dateipfad zu dem Originalbild und kann dieses bei Aufruf bereitstellen. In AudiVeris wird für Rasterbilder die Klasse *ByteProcessor* aus der Bibliothek *ij.process* verwendet. Zusätzlich hält *Picture* eine Reihe von Schlüsseln (*SourceKeys*) bereit, zu denen bestimmte Bilder zurückgegeben werden können. Darunter fallen z.B. der Schlüssel `GAUSSIAN`, der ein Gauss-gefiltertes Bild liefert. Diese Bilder sind nicht abgespeichert, sondern werden bei Anfrage errechnet.

Binärbilder werden als RunTable abgespeichert (s. 2.3.5). Diese werden als *.xml*-Dateien abgespeichert und bei Bedarf geladen.

Die Klasse „Glyph“

Eine Glyphe (engl. *glyph*) ist in AudiVeris ein Zeichen, das in keinem Zusammenhang zu Notenlinien oder -systemen steht. Es ist eine Ansammlung an Vordergrundpixeln, die ein Zeichen beinhalten, aber noch nicht zugeordnet wurden. Die Informationen sind als Rasterpunkte vorhanden.

Die Klasse „Inter“

Eine Inter ist die Interpretation einer Glyphe. Das was vorher als Zeichen ohne Beziehungen gespeichert war ist jetzt ein erkanntes Symbol im Zusammenhang der Takte, Notenlinien, -systemen und/oder Akkorden. Die bekannten Informationen sind die Position, die Art des Symbols und seine Beziehungen zu anderen Symbolen und den Notenlinien.

ByteProcessor

Die Klasse *ByteProcessor* befindet sich in der Bibliothek *ij.process* und erbt von der Klasse *ImageProcessor*. Sie ist ein 8-Bit-Bild und enthält Operation mit denen das Bild bearbeitet werden kann. Diese enthalten Histogrammbildung, Anwendung von LUTs (kurz für engl. *look up table*), Interpolation, Maxima, Minima und ROIs (Interessenregionen von engl. *regions of interest*).

2.2.4 Visualisierung

Die Ergebnisse einiger Rechenschritte werden als Bilder in AudiVeris angezeigt. Diese Bilder sind in der Mitte des Bildschirms angeordnet und können über Reiter (engl. *tabs*) ausgewählt werden. In der aktuellen Version wird das binarisierte Bild angezeigt und eine Visualisierung der gefundenen Daten. Für die Datenanzeige gibt es drei verschiedene Modi: `INPUT`, `INPUT_OUTPUT` und `OUTPUT`.

Der `INPUT`-Modus, sowie der `INPUT_OUTPUT`-Modus stellt die gefundenen Noten und Symbole überlagert über das Ausgangsbild dar. Die Glyphen sind klickbar und die Beziehungen zwischen Notenköpfen und -hälsen sind sichtbar. Die Benutzeroberfläche gibt wieder, welcher Bedeutung das Symbol zugeordnet wurde und mit welcher Wahrscheinlichkeit es anderen Klassen entspricht. Es können Symbole per Drag-And-Drop hinzugefügt werden.

Der `OUTPUT`-Modus zeigt nur die gefundenen Symbole als neues, sauberes Notenblatt.

Für jeden Tab, der auf der Oberfläche erscheinen soll, ist ein Eintrag in der Liste ***SheetTab*** angelegt.

2.3 Algorithmen

2.3.1 Das Otsu-Verfahren

Das Verfahren von Otsu ist eine Methode zur Segmentireung von Bildern. Da es ein Schwellwertverfahren ist, ist das Ergebnis standardmäßig ein Binärbild. Das heißt, dass es zwei Segmente gibt und das jedes Pixel exakt einem der beiden Segmente zugeordnet wird; es ist ein vollständiges und überdeckungsfreies Verfahren.

Sei t der Schwellwert (eng. *threshold*). Die Abbildung des Grauwertes g des betrachteten Pixels wird folgendermaßen bestimmt:

$$T_{global}(g) = \begin{cases} 0, & \text{falls } g < t \\ 1, & \text{falls } g \geq t \end{cases}$$

Die beiden Segmente werden hier mit *0* und *1* kodiert und symbolisieren meistens den Vordergrund und den Hintergrund. Es gibt auch Otsu-Verfahren mit mehreren Schwellwerten; diese haben dann kein Binärbild als Ergebnis.

Das globale Otsu-Verfahren ist einfach und schnell zu berechnen, ist allerdings anfällig auf Helligkeitsveränderung innerhalb des Bildes. Abhilfe schafft da ein regionales Segmentierungsverfahren, welches das Ausgangsbild in Interessenregionen einteilt. Für jede Region wird dann ein eigener Schwellwert berechnet und die Abbildung des Grauwerts ist wie folgt definiert:

$$T_{lokal}(x,y) = \begin{cases} 0, & \text{falls } g(x,y) < t_i \\ 1, & \text{falls } g(x,y) \geq t_i \end{cases} \quad \forall (x,y) \in RegionR_i$$

Diese Art der Regionalisierung kann allerdings zu Artefakten an den Übergängen zweier ROIs führen. Eine Möglichkeit das zu umgehen ist es, die regionalen Schwellwerte in einem Graph zu speichern und für jedes Pixel einen interpolierten Schwellwert zu benutzen.

Das Finden des Schwellwertes verhält sich bei regionaler und globaler Suche gleich. Für den zu betrachtenden Ausschnitt wird zunächst ein Histogramm angefertigt. Entlang des Spektrums dieses Histogramms werden alle Schwellwerte ausprobiert. Der Schwellwert teilt die Menge der Grauwerte in zwei Gruppen ein. Das Verfahren von Otsu sucht den Schwellwert der die Gruppen so teilt, dass die Streuung innerhalb der Gruppen möglichst gering ist und die Streuung zwischen den Gruppen möglichst hoch.

Die zwei Klassen K_0 und K_1 seien definiert als $K_0 = \{0, ..., t\}$ und $K_1 = \{t+1, ..., G\}$. Seien $\overline{g}_0$ und $\overline{g}_1$ die Mittelwerte der beiden Klassen, g der Grauwert des aktuellen Pixels und $p(g)$ die Auftrittswahrscheinlichkeit des Grauwerts. Die Wahrscheinlichkeit, dass ein Pixel der Klasse K_0 auftritt ist somit $P_0(t) = \sum_{g=0}^{t} p(g)$ und die Wahrscheinlichkeit, dass ein Pixel der Klasse K_1 auftritt ist $P_1(t) = \sum_{g=t+1}^{G} p(g) = 1 - P_0(t)$.

Die Varianzen sind:

$$\sigma_0^2(t) = \sum_{g=0}^{t} (g - \overline{g}_0)^2 p(g)$$

und

$$\sigma_1^2(t) = \sum_{g=t+1}^{G} (g - \overline{g}_1)^2 p(g)$$

Aus der Summe der beiden Varianzen errechnet sich die Varianz innerhalb der Klassen:

$$\sigma_{in}^2(t) = P_0(t) * \sigma_0^2(t) + P_1(t) * \sigma_1^2(t)$$

Die Varianz zwischen den beiden Klassen, wird wie folgt berechnet:

$$\sigma_{zw}^2(t) = P_0(t) * (\overline{g}_0 - \overline{g})^2 + P_1(t) * (\overline{g}_1 - \overline{g})^2$$

Der Quotient der beiden Varianzen lautet:

$$Q(t) = \frac{\sigma_{zw}^2(t)}{\sigma_{in}^2(t)}$$

Das Otsu-Verfahren liefert den Schwellwert, der die Varianz zwischen den Klassen maximiert und die Varianz innerhalb der Klassen minimiert. Der Schwellwert t für den der Quotient $Q(t)$ maximal ist, teilt also das Bild optimal, so dass das Binärbild den höchstmöglichen Informationsgehalt hat ([vgl. Jä12, S. 103]).

2.3.2 Sliding-Window

Die Sliding-Window-Methode (dt. gleitendes Fenster) iteriert über das Eingangsbild und teilt es in Abschnitte ein. Für diese Interessenregionen können dann beliebige Algorithmen angewandt werden. Regionen schließen einander nicht aus, d. h. benachbarte Gebiete können sich auch überlappen, wie der Name Sliding-Window erkennen lässt. Daher ist neben dem Radius bzw. Durchmesser die Überlappung ein essenzieller Parameter für den Algorithmus.

Im Gegensatz zu Filtern können die Fenster auch eine Seitenlänge von einer geraden Anzahl an Pixeln haben. Dies ist der Fall, da Filter einen Wert für das zentrale Pixel errechnen und dazu gegebenenfalls die Umgebung in Betracht ziehen. Bei Fenstern wird nicht ein Wert für das zentrale Pixel errechnet sondern der Wert wird in einen Graph geschrieben.

2.3.3 Hough-Transformation

Die *Hough-Transformation* ist ein Algorithmus zur Findung von geometrischen Formen in einem Binärbild. Für diese Arbeit ist die Linie als geometrische Form interessant. Eine Linie ist ein Teilabschnitt einer Geraden und Geraden lassen sich über zwei Parameter beschreiben: Den Winkel θ und den Abstand zum Bildursprung ρ.

Jedes Vordergrundpixel kann Teil einer solchen Gerade sein. Die *Hough-Transformation* nimmt für Vordergrundpixel den Wert 1 an. Diese Bildpunkte werden als Zeugen bezeichnet, denn jedes Pixel ist Zeuge der Geraden, auf der er liegt.

Für dieses Vordergrundpixel P werden für eine endliche Anzahl diskreter Winkel alle Geraden ausgerechnet, die durch den Punkt P gehen. Für jedes dieser Wertepaare (ρ, θ) ist P Zeuge([vgl. Pri15, S206]).

Im Parameterraum wird an der Stelle (ρ, θ) der Wert um eins erhöht. Wenn viele Pixel Teil einer Geraden sind, wird der Wert im Parameterraum an der entsprechenden Position für jeden Zeugen inkrementiert. Die Koordinaten mit den meisten Stimmen beschreiben die Geraden, die im Bild vorkommen.

2.3.4 K-Means-Clustering

Das *K-Means-Clustering* ist ein Algorithmus um Klumpungen und Häufungen (engl. *cluster*) in Wertemengen auszumachen. Als Voraussetzung für den Algorithmus muss die Anzahl k der Cluster festgelegt sein.

Beschreibe ein zweidimensionaler Vektor (x, y) einen Wert der Wertemenge. So erhalten alle Cluster ein initiales Zentrum$(\overline{x_i}, \overline{y_i})$, für das gilt: $0 \leq \overline{x_i} \leq max(X)$ und $0 \leq \overline{y_i} \leq max(Y)$. Die Zentren der Häufungen können zufällig oder gleichmäßig verteilt positioniert werden. In den folgenden Schritt werden alle Punkte dem ihnen am nächsten liegenden Zentrum zugeordnet. Das ist die erste Cluster-Einteilung. Im darauffolgenden Schritt werden die Zentren der Cluster angepasst; sie erhalten den Mittelwert aller Punkte, die zu der entsprechenden Gruppe gehören. Daraufhin werden die Punkte neu nach kürzesten Distanzen zugeordnet.

Diese Schritte können wiederholt werden, bis nur noch wenige Punkte das Cluster wechseln. Es gibt Anordnungen in denen ein perfektes Gleichgewicht nicht erreicht werden kann. Deswegen ist es unmöglich, den Algorithmus so zu formulieren, dass er erst endet wenn kein Punkt mehr das Cluster wechselt. Eine zweite Möglichkeit ist es, eine festgesetzte Anzahl an Wiederholungen machen.

Das *K-Means-Clustering* funktioniert auch bei mehrdimensionalen Wertemengen.

2.3.5 Lauflängenkodierung

Die Binärbilder werden in AudiVeris als *RunTable* gespeichert. Sie sind lauflängenkodiert (engl. *run length encoded*), d.h. dass nicht jeder Pixelwert abgespeichert wird, sondern die Anzahl der gleichen Pixel in Folge. Diese Lauflängenkodierung ist vertikal und horizontal möglich. Bei dieser Art der Datenspeicherung gibt es einen festgesetzten Wert, mit dem das Bild anfangen muss; z.B. *weiß* bzw. 1. Die erste Zahl im Speicher gibt dann an, wie

viele Pixel in Folge den gleichen Wert haben. Fängt das Bild mit einem schwarzen Pixel an, so muss an erster Stelle eine 0 stehen, denn es sind 0 weiße Pixel vorhanden. So kann die Zeichenkette 11111100100000 als sechsmal weiß, zweimal schwarz, einmal weiß und fünfmal schwarz oder $6, 2, 1, 5$ dargestellt werden.

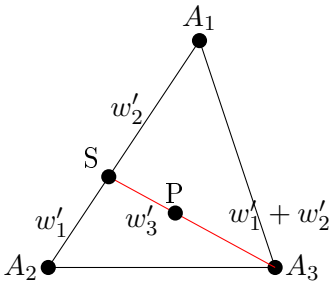
Mit dieser Methode können mit einem Byte beispielsweise 15 Pixel an Information gespeichert werden, während bei gängigen Graubildformaten ein Byte zur Speicherung eines Pixels notwendig wäre. Die Lauflängenkodierung verliert einen Teil ihrer Effizienz, wenn das Bild sehr viele Wechsel in Folge hat. Im schlechtesten Fall fängt das Bild mit einem schwarzen Pixel an und hat für jeden Nachfolger einen Wertewechsel. Dann wird für jedes Pixel eine Speichereinheit gebraucht und eine zusätzliche für die 0 am Anfang. Das ergibt jedoch keinen wesentlich höheren Speicheraufwand als das Bild in einem Raster abzuspeichern. Eine bessere Situation dagegen ist, wenn das Binärbild einfarbig weiß ist. Folglich ist die einzige Einschränkung die Größe der Zahlen, die eine Speichereinheit darstellen kann. Da jede neue Einheit einen Farbwechsel darstellt, muss auf jede voll ausgereizten Bitmenge eine 0 folgen. Im Idealfall erscheint jede der beiden Farben exakt so oft, wie es die Speichereinheit maximal darstellen kann, angefangen bei dem festgelegten Wert (in diesem Fall weiß).

2.3.6 Baryzentrische Koordinaten

Baryzentrische Koordinaten sind Zahlentripel, deren Zahlen die Gewichte der Eckpunkte eines Dreiecks $\Delta A_1, A_2, A_3$ darstellen.

Die Gewichte beschreiben einen Punkt P, der sich innerhalb des Dreiecks befindet und mit den Koordinaten (w_1, w_2, w_3) dargestellt wird. Die Eckpunkte des Dreiecks haben die Baryzentrischen Koordinaten $(1, 0, 0)(0, 1, 0)(0, 0, 1)$. Um die Baryzentrischen Koordinaten des Punktes P auszurechnen zieht man eine Gerade, die durch die Punkte A_3 und P geht und die Strecke $\overline{A_1A_2}$ in zwei Teile teilt. Da die Strecke $\overline{A_1A_2}$ die Länge 1 hat, ist das vorläufige Gewicht w_2' gleich dem Abstand vom Schnittpunkt S zum Eckpunkt A_1. Umgekehrt ist das Gewicht w_1' gleich $|\overline{SA_2}|$. Die Koordinate w_3' ist $|\overline{PS}|$.

Abbildung 2.3: Berechnung Baryzentrischer Koordinaten



[1]

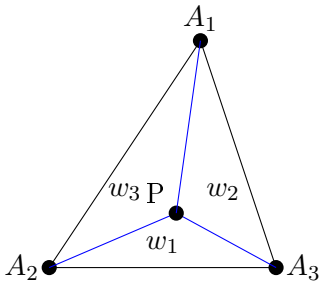
Die Voraussetzung für homogene Baryzentrische Koordinaten ist, dass $w_1+w_2+w_3 = 1$ gilt. Deswegen müssen die Werte mit $\mu = \frac{1}{w_1+w_2+w_3}$ homogenisiert werden.

$$\begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \mu * \begin{pmatrix} w_1' \\ w_2' \\ w_3' \end{pmatrix}$$

[1] Grafiken inspiriert von [Mat17]

Die Dreiecksflächen ΔA_2A_3P, ΔA_1A_3P und ΔA_1A_2P sind in ihren Verhältnissen proportional zu den Verhältnissen der Koordinaten.

Abbildung 2.4: Dreiecksflächen bei Baryzentrischen Koordinaten



Existieren diese Punkte in einem bereits festgelegten zweidimensionalen kartesischen Koordinatensystem, so können die Baryzentrischen Koordinaten wie folgt ausgerechnet werden:

$$w_1 = \frac{(A_{2y}-A_{3y})*(P_x-A_{3x})+(A_{3x}-A_{2x})*(P_y-A_{3y})}{(A_{2y}-A_{3y})*(A_{1x}-A_{3x})+(A_{3x}-A_{2x})*(A_{1y}-A_{3y})}$$

$$w_2 = \frac{(A_{3y}-A_{1y})*(P_x-A_{3x})+(A_{1x}-A_{3x})*(P_y-A_{3y})}{(A_{2y}-A_{3y})*(A_{1x}-A_{3x})+(A_{3x}-A_{2x})*(A_{1y}-A_{3y})}$$

$$w_3 = 1 - w_2 - w_1$$

Eine weitere Eigenschaft der Baryzentrischen Koordinaten ist, dass sie angeben können, ob ein Punkt innerhalb oder außerhalb des Dreiecks liegt. Wenn die Baryzentrischen Koordinaten nämlich aus kartesischen ausgerechnet werden, sind nur alle drei Koordinaten positiv, solange der Punkt innerhalb des Dreiecks liegt. Ist eines der Gewichte negativ, so liegt der Punkt außerhalb des Dreiecks.

3 Praktischer Teil

3.1 Einleitung

Dieses Kapitel beschreibt die Algorithmen und Datenstrukturen, die im Rahmen dieser Bachelorarbeit entstanden sind. Hierbei werden die relevanten Schritte der Bildvorverarbeitung näher erklärt. Grundlage für den Aufbau der Struktur lieferte die bereits vorhandene AudiVeris-Architektur. Der Beitrag, den diese Bachelorarbeit leistet, ist eine neue Binarisierung nach Otsu mithilfe des Sliding-Window-Verfahrens und eine Begradigung eventueller Krümmungen mithilfe der Hough-Transformation und des K-Means-Clustering.

3.2 Design

Die Implementierung und der Aufbau der neuen Schritte bedient sich der vorhandenen AudiVeris-Architektur. Die Handhabung der Bilddaten ist ebenfalls in die präsente Struktur eingearbeitet.

3.2.1 Ablauf

Um zusätzliche Operationen in die AudiVeris-Pipeline einzubauen, müssen sogenannte *Steps* eingefügt werden (vgl. 2.2.2 Pipeline in AudiVeris).

Dazu muss die Kennung des neuen Schrittes in die *Enumeration* ***Step*** eingefügt werden. Diese Liste enthält alle Schritte in der Reihenfolge, in der sie ausgeführt werden. Die ersten beiden Schritte `LOAD` und `BINARY` werden automatisch mit dem Öffnen eines neuen Buches ausgeführt. Außerdem ist eine neue Klasse nötig, die von der Klasse ***AbstractStep*** erbt. Der Konstruktor dieser neuen *Step*-Klasse wird ebenfalls in der Liste ***Step*** aufgerufen. Wie in Abschnitt 2.2.3 beschrieben, hat diese Klasse eine Methode `doit()`, in der der Ablauf der Operation gesteuert wird. Die Methode `displayUI()` ist verfügbar aber nicht notwendig; hierin können die Ergebnisse auf der Benutzeroberfläche dargestellt werden (s. 3.2.2 auf der nächsten Seite).

Der Hauptteil der Otsu-Segmentierung geschieht in der Klasse ***SWOtsuBinarizationFilter***. In der Methode `doit()` der Klasse ***SWOtsuBinaryStep*** wird die Methode `getSource()` der Klasse ***Picture*** aufgerufen. Wie in Abschnitt 2.2 beschrieben, werden die Bilder der Klasse ***Picture*** bei Anfrage ausgerechnet. Hierfür ist die oben erwähnte Klasse ***SWOtsuBinarizationFilter*** verantwortlich.

Diese Klasse berechnet, wie in Absatz 3.3 (Binarisierung) beschrieben, ein Binärbild und gibt es als ByteProcessor an die aufrufende Klasse ***Picture*** zurück.

3.2.2 Visualisierung

Zwischenergebnisse der Otsu-Binarisierung und der Ebnung benötigen *SheetTabs* um visualisiert zu werden (s. 2.2.4). Dafür werden Einträge in der *Enumeration* ***SheetTab*** angelegt: `SW_OTSU_BINARY_TAB` und `EVEN_TAB`.

3.3 Binarisierung

3.3.1 Aufbau

Der erste hinzugefügte Schritt der Bildvorverarbeitung ist die Binarisierung nach Otsu mit dem Sliding-Window-Verfahren.

Wie in Abschnitt 2.3.1 beschrieben ist die Otsu-Methode anfällig auf unebene Belichtung im Bild. Dies kann vor allem beim Scannen von gebundenen Büchern entstehen. Deswegen wird in dieser Arbeit ein regionaler Otsu-Algorithmus verwendet, der für die einzelnen Abschnitte im Bild lokal den idealen Schwellwert bestimmt. Die Regionen werden mit dem Sliding-Window-Operator festgelegt (s. 2.3.2). Für jede Region wird ein Knoten in einem Graph angelegt, welcher den Schwellwert und die Pixelkoordinaten enthält. Sind alle lokalen Schwellwerte bestimmt, wird für jedes Pixel des Ausgangsbilds ein Schwellwert interpoliert und somit das Binärpixel erstellt.

3.3.2 Sliding-Window-Parameter

Wie in Abschnitt 2.3.2 beschrieben sind die zwei wichtigen Parameter die Überlappung und der Radius bzw. Durchmesser. In dieser Anwendung wird der Radius verwendet, die Fenster haben also eine ungerade Seitenlänge und ein zentrales Pixel. Dieses zentrale Pixel wird auch als Koordinate für den Schwellwert-Knoten im Graph benutzt.

Um keine Pixel zu haben, für die eine Interpolation unmöglich wäre, ist der Sliding-Window-Operator in dieser Anwendung so angelegt, dass der Graph Knoten für die Bildecken hat. Die erste Reihe ist so gewählt, dass das zentrale Pixel des Fensters auf $y = 0$ liegt. Die Fenstergröße ist dabei angepasst. Normalerweise erstreckt sich das Fenster von dem zentrale Pixel ausgehend in alle vier Richtungen in Länge des Radius. An den Kanten des Bildes nimmt das Fenster nur den Bereich ein, der innerhalb der Bildgrenzen liegt. Damit sind allerdings nur die obere und linke Kante des Bildes abgedeckt (für $x = 0$ und $y = 0$). Um an den anderen beiden Extremen keine Lücken in der Interpolation zu erhalten, werden Fenster mit dem Mittelpunkt an der rechten und unteren Bildkante erstellt. Die Überlappung des letzten Fensters jeder Reihe mit seinen Nachbarn kann dabei von der sonstigen Überlappung abweichen.

3.3.3 Graph

Der Graph organisiert die Informationen in Knoten (engl. *nodes*). Diese sind als Elemente einer *LinkedList* vorhanden. Jeder Knoten hat eine Identifikationsnummer, die Koordinaten x und y und den Schwellwert, der für diese Punkt ermittelt wurde. Nachdem die Schwellwerte für jede ROI bestimmt wurden, werden die Knoten des Graphen in Dreiecke organisiert. Die Dreiecke sind ebenfalls Elemente einer *LinkedList*. Da die Knoten in einem regelmäßigen, geordneten Raster erstellt wurden, ist keine Positionsermittlung verbaut, sondern die Einteilung geschieht nach der Ordnung in der *LinkedList*. Nach diesem Schritt sind alle Knoten Teil mindestens eines Dreiecks und die Liste von Knoten wird nicht mehr verwendet.

Jedes Dreieck kann mithilfe Baryzentrischer Koordinaten für jedes gegebene Pixel bestimmen ob es innerhalb des Dreieck liegt (s. 2.3.6). Außerdem interpoliert es mit Baryzentrischen Koordinaten einen Schwellwert für das gegebene Pixel.

3.4 Ausgleich der Krümmung

In diesem Abschnitt wird der zweite Teil der Bachelorarbeit erklärt. Er beschäftigt sich mit dem Ausgleichen von Krümmungen im Ausgangsbild um die Notenextraktion zu ermöglichen oder zu verbessern.

3.4.1 Aufbau

Die Entkrümmung wird im Gegensatz zur Binarisierung in zwei Schritte aufgeteilt. Der erste Schritt ist für das Finden der Verzerrung zuständig während der zweite die Krümmung ausgleicht und ein begradigtes Bild für die Weiterverarbeitung schafft.

Für diese beiden Schritte wurden zwei neue Klassen in der AudiVeris-Architektur angelegt. Beide Klassen ordnen sich in die Pipeline ein und erben von der Klasse ***AbstractStep***.

Der Schritt, der die Krümmung im Bild erkennt heißt ***FindSheetCurvatureStep***. Er verwendet den *Hough-Line-Algorithmus* (s. 2.3.3 Hough-Transformation) um Linien im Binärbild zu finden. Danach wird die Menge der Linien gefiltert, um möglichst nur die Linien zu erhalten, die die Notenlinien beschreiben. Die Menge an Linien wird mit dem *K-Means-Clustering* in Gruppen eingeteilt (s. 2.3.4 K-Means-Clustering). Dies geschieht, um einer ungleichen Krümmung entlang der Höhe des Blattes vorzubeugen. Für jede dieser *Cluster* wird eine *Funktion* erstellt, die für jeden X-Wert die mittlere Steigung der Linien enthält. Diese Funktionen werden an die Klasse ***Picture*** übergeben und dort für den nächsten Schritt gespeichert.

Der nächste Schritt begradigt das Binärbild nach der im vorherigen Schritt erkannten Krümmung. Er heißt ***EvenStep***. Dieser Schritt lädt die *Funktionen* und nutzt diese um das Binärbild zu begradigen (Bsp. einer Funktion s. Abb. 3.5). Hierbei wird für jedes Pixel im Bild die Steigung aus den Funktionen gelesen, und das Pixel um diesen Wert in Y-Richtung verschoben.

3.4.2 Function

Die Krümmung des Notenblatts wird in einer sogenannten `function` beschrieben. Eine `function` ist ein *Array* aus Dezimalzahlen, die die kumulierte Steigung beschreiben. Dieses Array ist um eine Stelle länger als die Bildbreite. Auf der letzten Stelle steht die Höhe in Pixeln. Diese wird aus dem Durchschnitt der Y-Koordinaten aller Linien gewonnen, die zu dem *Cluster* der Funktion gehören.

Für jede Stelle in der Funktion werden alle Linien, die an der entsprechenden X-Koordinate präsent sind in Betracht gezogen. Die Steigungen der Linien werden gemittelt und die lokale Steigung wird auf die des Vorgängers addiert.

3.4.3 FindSheetCurvatureStep

Der Schritt zur Findung der Krümmung ist das erste der beiden Mittel zur Begradigung des Bildes. Zuerst wird das Binärbild geladen und invertiert. Die Inversion ist notwendig, da in AudiVeris die Vordergrundpixel mit schwarz (also binär 0) kodiert werden. In dem

Hough-Line-Algorithmus der Bibliothek *OpenCV* werden Vordergrundpixel allerdings mit weiß (binär 1) kodiert.

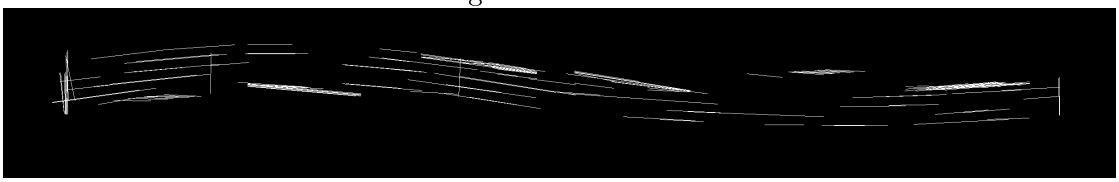
Dann wird die Methode *HoughLinesP* aus *OpenCV* aufgerufen. Im Gegensatz zu dem in Abschnitt 2.3.3 beschriebenen Paaren aus Winkel und Abstand liefert diese Funktion eine Menge an Linien. Diese Linien sind definiert durch ihren Anfangs- und Endpunkt. Somit ist *HoughLinesP* eine Erweiterung der eigentlichen Hough-Line-Findung, die die gefundenen Geraden zusätzlich begrenzt.

Neben dem invertierten Ausgangsbild und einer leeren Matrix für die gefundenen Linien werden der Methode einige weitere Parameter übergeben. Die Schrittweite, in der das Bild abgesucht wird, ist auf ein Pixel gesetzt. Für die Winkel, die für jeden Pixel als Linie vermutet werden, ist die Schrittweite ein Grad in Radiant. Die Länge, die eine Reihe Vordergrundpixel mindestens haben muss, um als Linie anerkannt zu werden beträgt ein Zweiunddreißigstel der Bildbreite. Die maximale Länge, die Lücken in dieser Linie haben dürfen ist 15 Pixel.

Abbildung 3.1: Notenzeile mit Krümmung und Gradient

Die Menge der gefundenen Linien enthält allerdings auch einige Elemente, die zur Findung der Krümmung hinderlich sind. Wie auf der Abbildung 3.2 zu sehen ist, sind Taktstriche und Notenbalken ebenfalls Ursprünge gefundener Linien.

Abbildung 3.2: Gefundene Linien



Diese Linien werden mit zwei Funktionen entfernt. Um die Taktstriche und Notenhälse auszuschließen, werden alle Linien aus der Menge gelöscht, die zur Y-Achse in einem Winkel von $-45°$ bis $45°$ stehen. So bleiben die Linien übrig, die in Abb. 3.3 zu sehen sind.

Die Linien, die von den Notenbalken stammen, haben eine ähnliche Richtung wie die Notenlinien. Daher wird eine andere Strategie angewendet, um diese zu ermitteln. Da der Notenbalken breiter ist, als die meist nur einen Pixel starke Notenlinie, wird nach Häufungen mehrerer Linien an den gleichen Positionen gesucht. Einige der Linien, die die Richtung der Notenlinien beschreiben, haben allerdings auch Berührungspunkte,

und zwar wenn der Endpunkte einer Linie nahe dem Anfangspunkte der nächsten liegt. Deswegen werden in der Funktion `deleteBeams()` (von engl. *beam*, bedeutet Balken) alle Linien entfernt, deren Anfangspunkte in einem Abstand von zehn Pixeln oder geringer liegen. Das Ergebnis ist in Abb. 3.4 zu sehen.

Da nun größtenteils nur Linien übrig sind, die tatsächlich die Notenzeilen beschreiben, kann die Seitenkrümmung ermittelt werden. Für jedes x werden die Steigungen aller Linien addiert, die durch diese X-Koordinate gehen. Diese Summe wird durch die Anzahl der relevanten Linien geteilt, um das arithmetische Mittel zu erhalten. Das Mittel wird dann auf die Steigung des vorhergehenden X-Werts addiert, um die absolute Verschiebung in Y-Richtung zu erhalten. Eine positive Zahl drückt dabei aus, dass die Krümmung sich nach oben bewegt und zum Ausgleich wieder nach unten verschoben werden muss. Wie in Absatz 3.4.2 auf Seite 23 beschrieben, steht in der letzten Stelle der Funktion die mittlere Y-Position aller Linien des aktuellen *Clusters*.

Um grobe Ausreißer und scharfe Richtungsänderungen zu verhindern wird die Funktion mit einem eindimensionalen Gauß-Filter geglättet. Dieser hat einen Radius von 2 und damit die Form $\{1, 4, 6, 4, 1\}$.

In Abb. 3.5 ist die gefundene Verschiebungsfunktion zu sehen.

3.4.4 EvenStep

Der Schritt ***EvenStep*** zur Ebnung eventueller Bögen lädt zunächst die Funktionen und das Binärbild aus ***Picture***.

Alle Funktionen, die Steigungen mit ungültigen Werten enthalten, werden verworfen.

Jedes Pixel des Binärbildes muss um einen Wert in Y-Richtung verschoben werden. Da aus jedem *Cluster* von Linien eine eigene Funktion ermittelt wurde, wird zunächst ausgemacht, welche Funktionen für das Pixel relevant sind.

Für alle Pixel, die zwischen zwei Funktionen liegen, wird die Y-Verschiebung interpoliert. Alle Pixel, die unterhalb des Y-Wertes der untersten Funktion liegen wird kein Wert interpoliert. Stattdessen wird die Steigung der untersten Funktion genommen. Äquivalent verhält es sich für Y-Werte, die höher als die oberste Funktion liegen.

Die Interpolation basiert lediglich auf den Abständen in Y-Richtung. Sei der Abstand der Mittel der beiden Funktionen $|\mu_i - \mu_{i-1}|$. Der Abstand zwischen dem aktuellen Punkt und der ersten Funktion ist dann $|y - \mu_{i-1}|$ und das Gewicht mit welchem der erste

Abbildung 3.3: Linien ohne vertikale Linien

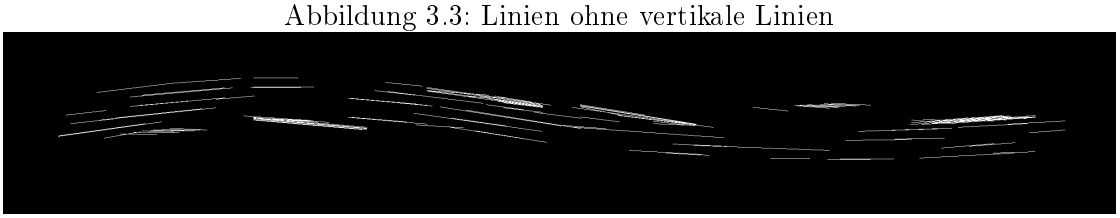
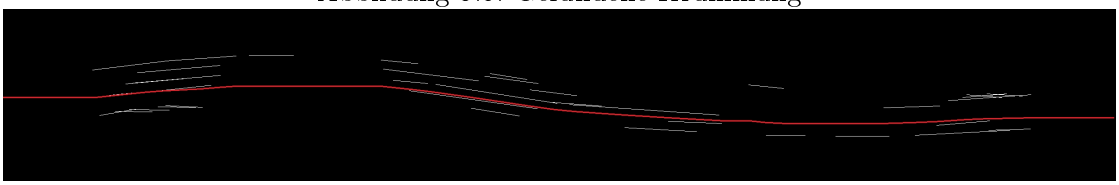


Abbildung 3.4: Linien ohne stark gehäufte Linien

Abbildung 3.5: Gefundene Krümmung



Versatz multipliziert wird $w_{i-1} = \frac{|\mu_i - \mu_{i-1}|}{|y - \mu_{i-1}|}$. Das Gewicht für den Versatz aus der anderen Funktion wird mit $w_i = 1 - w_{i-1}$ ausgerechnet.

Der Versatz, um den das aktuelle Pixel verschoben wird, ergibt sich aus den Steigungen aus den beiden Funktionen, multipliziert mit ihren Gewichten. Das geebnete Bild wird dann als *RunTable* gespeichert und ersetzt das vorherige Binärbild für alle weiteren Operationen.

3.5 Liste neuer Klassen

Diese Klassen wurde im Zuge der Arbeit AudiVeris hinzugefügt:

- ***SWOtsuBinaryStep***
- ***SheetCurvatureStep***
- ***EvenStep***
- ***SWOtsuBinarizationFilter***

In der Klasse ***Picture*** wurde die Prozedure `swoBinarized` hinzugefügt. Diese Methode ruft die Klasse ***SWOtsuBinarizationFilter*** auf.

Abbildung 3.6: Geebnete Notenzeile

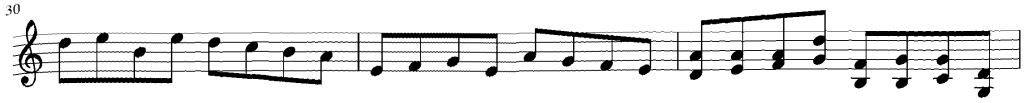
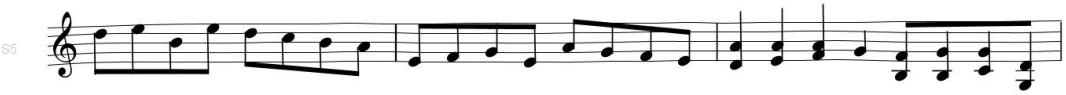


Abbildung 3.7: Erkannte Notenzeile



4 Experimente

4.1 Aufbau

Zur Überprüfung der Nützlichkeit der Bildvorverarbeitung wird die Notenerkennung in AudiVeris in drei verschiedenen Versionen getestet. Zuerst mit der unveränderten Ausgabe, dann mit der regionalen Binarisierung nach Otsu und schließlich mit der Binarisierung und der Entkrümmung. Die unveränderte Ausgabe wird im folgenden auch *Originalversion* genannt, die Ausgabe mit der regionalen Segmentierung nach Otsu wird *SWO-Version* genannt (Abk. von *Sliding-Window-Otsu*). Die Ausführung, welche die Begradigung der Seitenkrümmung beinhaltet wird *Ebnung* genannt.

Um zu testen, wie gut Noten erkannt werden können, werden verschiedene verunreinigte Notenblätter benutzt.

Zusätzlich wird die Zeit für die Teilschritte der Binarisierungsmethoden gemessen und verglichen. Für die Kurvendetektion und die Ebnung werden ebenfalls die Laufzeiten gemessen.

4.1.1 Bilddatenbank

Die Bilddatenbank besteht aus Musiknotenblättern, die mit dem Programm MuseScore 2 erstellt wurden. In diesen Notenblättern sind ganze, halbe, Viertel-, Achtel- und Sechzehntelnoten enthalten. Dieses beispielhafte Notenblatt wurde durch verschiedene Algorithmen verunreinigt und verzerrt.

Darunter fällt das *Salz-und-Pfeffer-Rauschen*, welches für jeden Pixel einen Zufallswert auswählt, der entweder auf weiß oder schwarz fällt. Dieser Effekt soll den Notenerkennungsalgorithmus auf Robustheit gegen Körnung in der Fotoqualität und generelle Unreinheiten überprüfen.

Eine weitere Verzerrung ist die sogenannte *Buchkrümmung*. Hierbei wird das Bild mithilfe einer Sinuskurve in Y-Richtung verschoben, um eine Krümmung ähnlich einem gebundenen Buch zu simulieren. In der linken Hälfte des Bildes werden die Pixel in einem Bogen nach oben verschoben, in der rechten Hälfte dagegen nach unten. Die Amplitude des rechten Bogens ist halb so groß, wie die des linken Bogens. Ein weiterer Aspekt der Buchkrümmung ist ein Schwarz-Weiß-Verlauf, der von links aus ein Viertel des Bildes einnimmt.

Des weiteren wird eine *Kombination* der beiden vorhergehenden Effekte getestet. Eine Buchkrümmung mit zusätzlichem *Salz-und-Pfeffer-Rauschen*.

Als vierte Verunreinigung wird ein *Grauwertverlauf* von links ausgehend über das gesamte Bild gezogen. Diese soll die ungleiche Belichtung simulieren und das Erkennungsverfahren dahingehend prüfen.

Jeder dieser Effekte wird mit einem gewissen Grad auf das Ausgangsbild angewendet. So wird beim *Salz-und-Pfeffer-Rauschen* beispielsweise das zufällige Pixel zu einem gegebenen Prozentsatz mit dem Notenblatt verrechnet. Dadurch werden die Bildverarbei-

tungsschritte mit ansteigenden Graden von Verunreinigungen getestet. In den folgenden Experimenten werden Bilder verwendet, die zu 5, 10, 15, 20, 30 und 50 % mit den oben beschriebenen Effekten verändert wurden.

4.1.2 Test-Konfiguration

Die verzerrten und verunreinigten Bilder werden in drei verschiedenen Ausgaben des Programmes getestet und die Anzahl der erkannten Noten wird gezählt.

Die Zeit wird durch eine bereits in AudiVeris verbaute Stoppuhr genommen.

4.2 Evaluationskriterien

Zum Testen der Bildvorverarbeitung wird überprüft, ob mehr Noten erkannt werden können, als ohne die Verarbeitungsschritte. Dazu wird unterschieden, wie viele Notenköpfe, -hälse und -balken gefunden wurden, und ob alle erkannten Noten richtig ihrem Notenwert zugeordnet wurden. Die Tonhöhe wird in diesem Fall vernachlässigt. Wenn nur ein Teil der Note richtig *erkannt* wird, der Rest aber nicht, wird diese Note als nicht richtig *erkannt* eingestuft. Findet das Programm beispielsweise einen Notenkopf und den dazugehörigen Hals, aber nicht den Balken, so gilt die Note als nicht erkannt. Schließlich wurde eine Achtelnote als Viertelnote interpretiert; der falsche Notenwert ist inkorrekt. Dennoch gelten der Notenkopf und der -hals als *gefunden*. Außerdem wird getestet, ob Bilder, die von der Original-Version als ungültig eingestuft wurden, durch die Bildvorverarbeitung erkennbar werden.

Der Test zählt, wie viele der drei Elemente *Notenkopf, -hals und -balken* gefunden und wie viele der einzelnen Notentypen *ganze Note, halbe Note, Viertelnote, Achtelnote und Sechzehntelnote* erkannt wurden.

4.2.1 Zuverlässigkeit

In der Entwicklungsphase dieser Arbeit konnte eine Schwierigkeit mit der `cluster`-Funktion aus der Bibliothek *JavaML* festgestellt werden. Der zweite Teil dieses Kapitels beschäftigt sich mit der Robustheit der Begradigung. In den Tests wird überprüft, wie oft der Schritt ***SheetCurvatureStep*** erfolgreich ausgeführt werden kann und wie oft die Methode `cluster` nach 120 Sekunden ein *Time-Out* auslöst.

4.3 Experimente

Die Experimente in diesem Kapitel testen die Zuverlässigkeit, Geschwindigkeit und Genauigkeit der Verarbeitungsschritte.

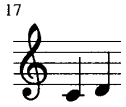
4.3.1 Salz-und-Pfeffer-Rauschen

Im Falle des Salz-und-Pfeffer-Rauschens schneidet der regionale Otsu-Algorithmus besser ab, als die Binarisierung der Original-Version. Bei der originalen Segmentierung bleibt ab einem Prozentsatz von 20 eine Körnung in Nähe der Symbole übrig, die die Otsu-Binarisierung eliminieren kann (s. Abbildungen 4.1 und 4.2).

Für alle Rauschbilder hat das SWO-Verfahren $83,3\%$ aller Noten gefunden und ebenso $83,3\%$ richtig zugeordnet. Das Bild mit 50-prozentigem Rauschen wurde als ungültig gewertet und konnte von AudiVeris nicht weiter verarbeitet werden.

Abbildung 4.1: Original-Version, 20-prozentiges Rauschen

Abbildung 4.2: SWO-Version, 20-prozentiges Rauschen



Im Gegensatz dazu hat die ursprüngliche Binarisierung eine Erfolgsquote von 50% gefundenen und erkannten Noten. Bei dieser Version konnten das 30- und das 50-prozentig verrauschte Bild nicht weiterverarbeitet werden und galten als ungültig. Das Bild mit einer Rauschsichtbarkeit von 20 % (s. Abb. 4.1) wurde bis zum Ende bearbeitet, allerdings konnten keine Informationen extrahiert werden.

Bei der Ebnung-Version verhielt es sich ähnlich wie bei der SWO-Version, da beide Ausgaben den gleichen Binarisierungsalgorithmus nutzen. Jedoch fügt der Begradigungsschritt Artefakte ein, die ein vorher ebenes Bild unleserlicher machen. Darauf wird ausführlicher in Abschnitt 4.3.2 eingegangen. Deshalb erreichte diese Version eine Findungsquote von $77,3\%$ und eine Zuordnungsgenauigkeit von $72,7\%$.

Für Rauschbilder kann also der Otsu-Algorithmus ein Notenblatt zu 100% richtig erkennen, wo die Original-Segmentierung keine einzige Note gefunden hat. Außerdem wird ein vorher ungültiges Bild brauchbar gemacht und ebenfalls zu 100% richtig erkannt.

4.3.2 Grauwertverlauf

Der Grauwertverlauf stellte für beide Version der Segmentierung ein geringes Hindernis dar. Selbst im stärksten Grade (50% Sichtbarkeit) konnten beide Segmentierungsverfahren das Bild nahezu fehlerlos zweiteilen. Probleme, die beim Otsu-Algorithmus auftauchten, sind Randzonen, in denen keine Vordergrundpixel vorhanden waren. In diesen Fällen umschließt die ROI eine Fläche auf der nur Pixel sind, die als Hintergrund interpretiert werden sollen. Jedoch sind dort auch mehrere Grauwerte vorhanden, welche dann binarisiert werden. Das Ergebnis ist in Abbildung 4.3 zu sehen.

Abbildung 4.3: Randartefakt

So erkannte bei 50-prozentigem Grauwertverlauf die Otsu-Version 206 von 209 Noten richtig $(98,6\%)$, während die originale Binarisierung alle Noten erkannte.

Die beiden Segmentierungsverfahren unterscheiden sich also nur gering. Allerdings ist in diesem Abschnitt sichtbar, welchen Einfluss die Begradigung auf ein Bild hat, das keine Begradigung benötigt. Viele Linien beeinflussen die Krümmungsfindung und Artefakte sind keine Seltenheit. Trotz des Schrittes, der die Linien entfernen soll, die durch die Notenbalken entstehen (vgl. 3.4.3), beeinflussen Fremdlinien die Biegungsfunktion. Bei einem in Bezug auf die Krümmung unveränderten Bild, müsste diese Funktion eine Gerade parallel zur X-Achse sein. Wenn diese jedoch abweicht, wird das Bild anstatt begradigt zu werden verkrümmt. Jeder Schritt, der das Bildmaterial zusätzlich verändern, stellt ein Risiko für die Notenextraktion dar.

So kann man erkennen, dass die Ebnung-Version deutlich schlechter mit einem Grauverlauf umgehen kann als die beiden anderen Versionen.

Tabelle 4.1: Findungs- und Erkennungsrate

Version	**gefunden**	**erkannt**
Original	100,0%	100,0%
SWO	99,9%	99,8%
Ebnung	71,6%	61,8%

4.3.3 Krümmung

Bei den gekrümmten Testbildern ist die Anzahl an extrahierten Noten gering. Die Original-Version fand durchschnittlich nur 10,7% aller Notenköpfe, die SWO-Version 11,4%. Der Anteil der richtig zugeordneten Noten liegt für die Original-Version bei 9,4% und für die Otsu-Version bei 9,5%.

Die Ebnung-Version schafft es, 32,1% aller Notenköpfe zu finden und 23,8% aller Noten richtig zu erkennen.

4.3.4 Krümmung und Rauschen

Die Kombination von *Salz-und-Pfeffer-Rauschen* und der *Krümmung* liefert ähnliche Resultate wie die *Krümmung* allein. Auch hier übertrifft die Ebnung-Version mit 19,6% gefundenen Noten die anderen beiden Ausführungen der Software (4,2% für die Original-Version und 10,3% für die SWO-Version).

Allerdings ist keine der Anwendungen besonders zuverlässig, wenn es ein verrauschtes und verbogenes Bild zu analysieren gilt. In etwa der Hälfte der Fälle konnte die Notenfindung nicht beendet werden, weil AudiVeris keine Notenlinien finden konnte oder die Linien keinem System zuordnen konnte. Die Original-Version konnte nur zwei der sechs Blätter abschließen, die Ebnung-Version drei. Am weitesten vorne lag die SWO-Version mit vier von sechs beendeten Notenblättern.

Effektivität

Der Anteil der gefundenen und erkannten Noten beträgt bei den beiden neuen Versionen (SWO und Ebnung) ungefähr 50%, während der Anteil der Original-Version sich auf ca. 40% beläuft. Das Diagramm 4.4 zeigt den Prozentsatz der richtig erkannt und zugeordneten Noten nach Notenwert.

Abbildung 4.4: Anzahl der gefundenen Notentypen im Durchschnitt

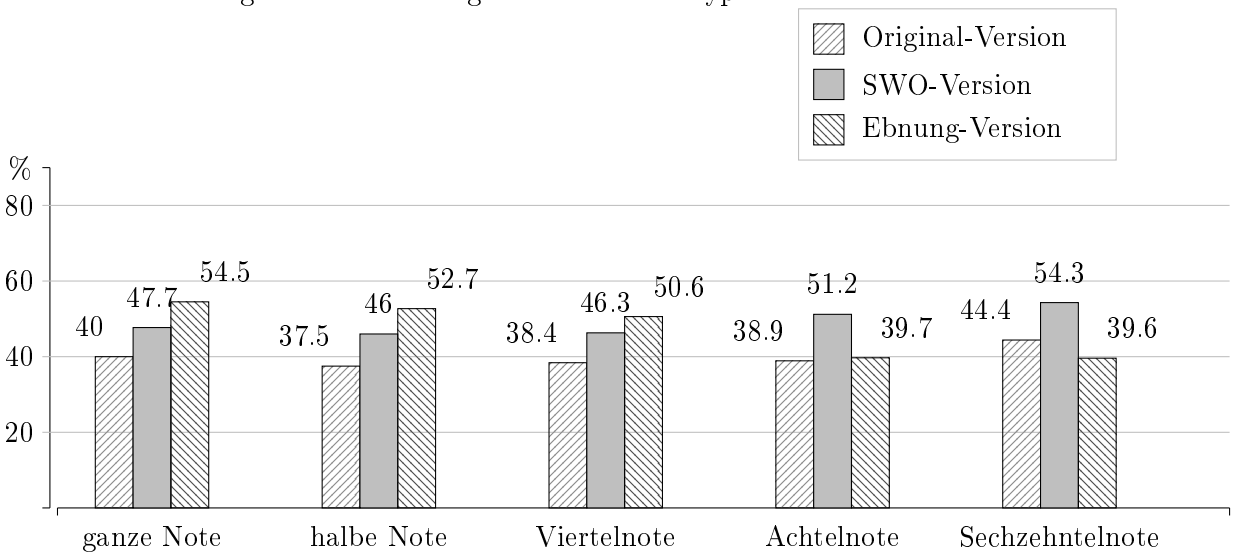
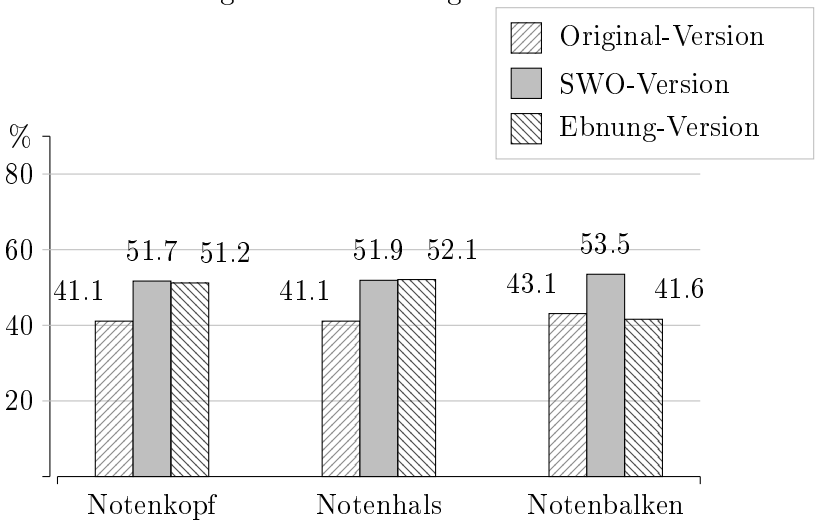


Abbildung 4.5: Anzahl der gefundenen Notenelemente im Durchschnitt



Es ist erkennbar, dass die Entkrümmung in einigen Fällen einen Vorteil bringt und es der Software ermöglicht mehr Noten richtig zu erkennen. In anderen Fällen wird das Erkennen sogar gehindert, so dass die unveränderte Version von AudiVeris mehr Sechzehntelnoten erkenne konnte, als die Ebnung-Version. Dies wird auch in dem Diagramm 4.5 sichtbar, wo besagte Ausgabe die wenigsten Notenbalken zuordnen konnte. Wann immer ein Notenbalken nicht gefunden wird, können die dazugehörigen Noten nicht als der richtige Notenwert erkannt werden.

Eine zweite Beobachtung, die aus den Diagrammen zu schließen ist, ist dass die *Sliding-Window-Otsu-Binarisierung* konstant bessere Werte liefert, als die ursprüngliche.

4.3.5 Performanz

Die Binarisierung, die in dieser Arbeit eingebaut wurde, brauchte länger als die vorherige. Im Durchschnitt rechnete die SWO-Binarisierung 3177 Millisekunden länger als die der Original-Version.

Tabelle 4.2: Zeitaufwand der Binarisierungen [in ms](Auszug)

Version	**binarisieren**	**RunTable erstellen**	**Gesamt**
Original	1230	163	1393
SWO	5958	150	6109
Original	1211	176	1387
SWO	5649	146	5796
Original	1170	255	1425
SWO	4741	121	4863
Original	1207	278	1486
SWO	4878	158	5085

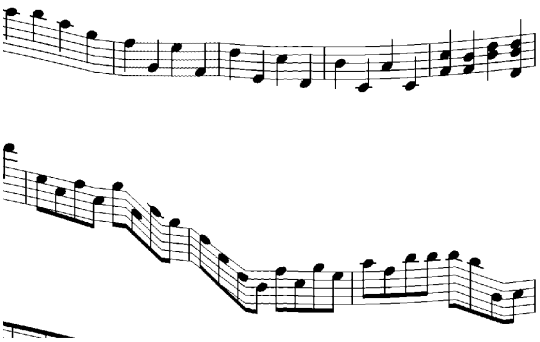
4.3.6 Zuverlässigkeit

Die Binarisierung nach Otsu mit der Sliding-Window-Methode zu Einteilung der Regionen erwies sich in den Tests als zuverlässig. Das Programm stürzte nicht ab und beendete die Notensuche in allen Fällen, in denen die originale Binarisierung diese beendete. Hinzu kommt, dass die neue Binarisierung es dem Programm ermöglichte die Suche bei höheren Rauschgraden als mit der ursprünglichen Segmentierung auszuführen. Bei Verunreinigungsgraden, bei denen die Ursprungsversion den Vorgang abbrechen musste, konnten mithilfe der neuen Binarisierung sogar alle Noten gefunden werden.

Im zweiten Test, der sich der Stabilität der Krümmungssuche widmet, konnte die Ebnung-Variante der Software 36% der Versuche mit einem produzierten Ergebnis abschließen. In 12% der Fälle musste die Begradigung abgebrochen werden, weil das *Clustering* nach einer Rechenzeit von 120 Sekunden keine Rückmeldung gab. In den restlichen 52% vollendeten ***SheetCurvatureStep*** und ***EvenStep*** ihre Aufgabe, AudiVeris brach aber an einer späteren Stelle ab, weil wichtige Daten nicht extrahiert werden konnten. Durch die Krümmung des Blattes konnten die Zeilenköpfe inklusive der Notenschlüssel in einigen Fällen nicht gefunden werden und verursachten einen Abbruch. In anderen Fällen brachte die Unmöglichkeit, die Notenzeilen zu Systemen zu organisieren, das Programm zum Stillstand.

Da viele Linien die Krümmungsfindung beeinflussen, geschah es auch, dass die Ebnung genau das Gegenteil ihres Ziels bezweckte. Ein Beispiel davon ist in Abb. 4.6 zu sehen.

Abbildung 4.6: Versagen der Ebnung



5 Fazit

5.1 Zusammenfassung

In dieser Arbeit wurde die Bildvorverarbeitung in AudiVeris dahingehend verbessert, dass eine Binarisierung eingebaut wurde, die Bilder bis in hohe Rauschgrade nutzbar machen kann. Dazu wurde das Verfahren nach Otsu genutzt, um lokale Schwellwerte zu finden und das Sliding-Window-Verfahren, um das Bild in Regionen aufzuteilen. Außerdem wurde der Vorverarbeitung mit der Ebnung ein neuer Aspekt hinzugefügt. Mit der Hough-Transformation zur Linienfindung und dem K-Means-Algorithmus zur Findung von Clustern konnte dieser Schritt situationsbedingt die Ergebnisse verbessern.

Vorher wurde das Bild durch ein Rauschen mit 20-prozentiger Deckkraft bereits unleserlich gemacht.

Die Experimente zeigen, dass die Binarisierung nach Otsu mit zusätzlicher Regionalisierung generell besser abschneidet als die bisher verbaute Binarisierung. Selbst wenn sie im Schnitt drei Sekunden länger braucht, kann es dennoch zum Vorteil sein, denn die SWO-Binarisierung kann Rauschen mit bis zu 30-prozentiger Deckkraft eliminieren.

Wenn es zur Begradigung von Biegungen im Bildmaterial kommt, sind die Vorteile fallabhängig. In Bildern mit Krümmung, in denen die anderen Versionen sehr schlecht abschnitten, fand das Programm mit Krümmungsausgleich einige Noten mehr. Jedoch konnte kein Notenblatt zu besonders hohen Prozentsätzen (70% oder höher) gelesen werden. Das Clustering verursacht einige Timeouts. Zudem stört ein falsch zusammengefasstes Cluster die Bildstruktur erheblich und macht einen Bereich unkenntlich, der bis zu zwei Systeme umfassen kann. Vor allem aber führte der Begradigungsschritt bei Seiten, die keine Begradigung nötig hatten, neue Schwierigkeiten für die Notenerkennung ein.

Durch die Experimente wurde klar, dass der Schritt der Entkrümmung einen größeren Nutzen haben kann, wenn er zuschaltbar ist in Situationen, die nach ihm verlangen.

Die Schwierigkeiten dieser Arbeit lagen darin, eine fremde und komplexe Projektstruktur zu durchsuchen und zu verstehen. Zusätzliche Schritte wurden nach dem Vorbild der vorhandenen Architekturen verbaut um eine Übereinstimmung zu gewährleisten. Die Daten wurden nach der Verarbeitung in arbeitsspeicher-effizienten Weisen gelagert.

Zur Entwicklung und Fehlersuche werden die Zwischenergebnisse der Krümmungsfindung und der Ebnung visualisiert. Die Verschiebungsfunktionen werden dem Binärbild überlagert, damit der Benutzer sehen kann, ob sie der Krümmung der Notenlinien folgen.

5.2 Ausblick

In den Experimenten kam zum Vorschein, dass die Binarisierung nach Otsu eine Verbesserung der Notenerkennung zur Folge hat. Der Ausgleich der Krümmung dagegen brachte nicht immer Vorteile. Dies kann in Zukunft positiv genutzt werden indem die beiden

verantwortlichen Schritte zuschaltbar werden. Dies ist innerhalb der AudiVeris-Struktur über globale Variablen möglich, die man in der Benutzeroberfläche setzen kann. Außerdem ist durch die Teilung des Schrittes in Funktionsdetektion und Funktionsanwendung ein Kontrollabschnitt von Seiten des Benutzers möglich. Ausreißende Funktionen können in Zukunft vom Bediener gelöscht oder bearbeitet werden.

Außerdem ist eine automatische Selektion von Ausreißern vorstellbar, bevor die Funktionen erst angelegt werden. Dazu wäre eine Prüfung der Linien anhand ihrer Steigung möglich. Dies wäre zusätzlich zu der Methode, die vertikale Linien entfernt denkbar, da dies bereits ein einfaches Verfahren ist, eindeutige Ausreißer zu entfernen und somit die Varianz nicht unnötig zu erhöhen.

Weiterhin ist es sinnvoll zukünftig einen Schritt einzufügen, der die Anzahl der zu suchenden Cluster ermittelt. Da bei Notenblätter die Notensysteme natürliche Cluster bilden, ist es denkbar, das Binärbild horizontal zu summieren und auf dem eindimensionalen Ergebnisbild nach Schwellwerten oder Wasserscheiden zu suchen, um die Anzahl an Notensystemen zu finden.

6 Anhang

6.1 Anteil der erkannten Noten

[1]

[1]SUP = Salz-und-Pfeffer-Rauschen, BK = Buchkrümmung, BSUP = Kombination aus Buchkrümmung und Salz-und-Pfeffer-Rauschen, GRA = Grauwertverlauf

Effekt	Version	ganze N.	halbe N.	Vierteln.	Achteln.	Sechzehnteln.
Kein	Original	100,0%	100,0%	100,0%	100,0%	100,0%
	SWO	100,0%	100,0%	100,0%	100,0%	100,0%
	Ebnung	100,0%	100,0%	81,1%	43,1%	76,4%
SUP	Original	50,0%	50,0%	50,0%	50,0%	50,0%
	SWO	85,7%	85,7%	85,7%	85,7%	85,7%
	Ebnung	81,4%	85,7%	82,2%	72,0%	62,5%
BK	Original	10,0%	0,0%	3,6%	5,6%	18,5%
	SWO	5,0%	0,0%	0,0%	9,3%	17,6%
	Ebnung	41,7%	30,6%	26,6%	11,8%	30,3%
BSUP	Original	0,0%	0,0%	0,0%	0,0%	9,3%
	SWO	0,0%	0,0%	0,0%	10,0%	13,9%
	Ebnung	13,3%	11,1%	15,8%	19,4%	13,9%
GRA	Original	100,0%	100,0%	100,0%	100,0%	100,0%
	SWO	100,0%	98,1%	99,5%	100,0%	100,0%
	Ebnung	81,7%	83,3%	77,9%	55,6%	51,6%
Gesamt	Original	40,0%	37,5%	38,4%	38,9%	44,4%
	SWO	47,7%	46,0%	46,3%	51,2%	54,3%
	Ebnung	54,5%	52,7%	50,6%	39,7%	39,6%

6.2 Beispiele einiger Testbilder

Abbildung 6.1: Ausgangsbild für die Experimente

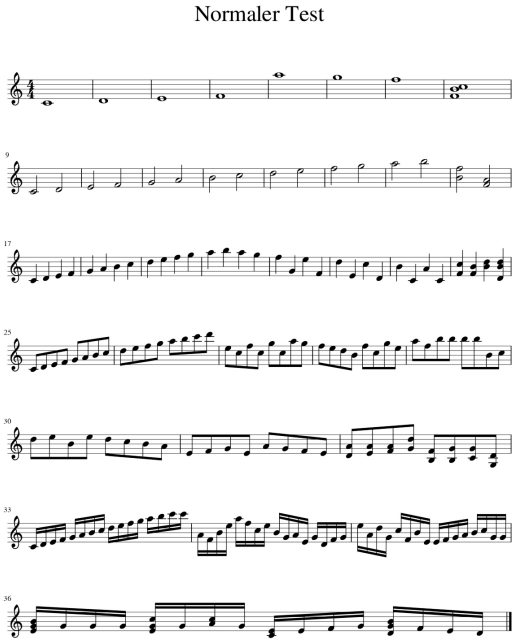


Abbildung 6.2: 15-prozentige Buchkrümmung

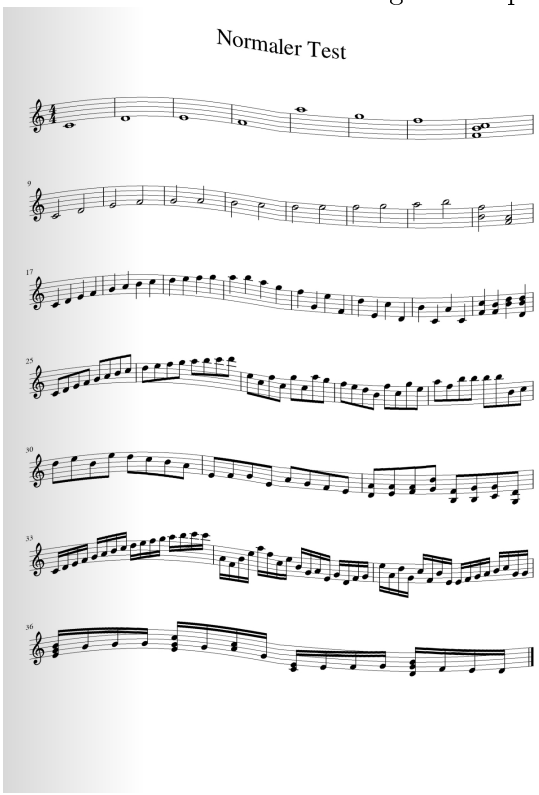


Abbildung 6.3: 20-prozentiges Salz-und-Pfeffer-Rauschen

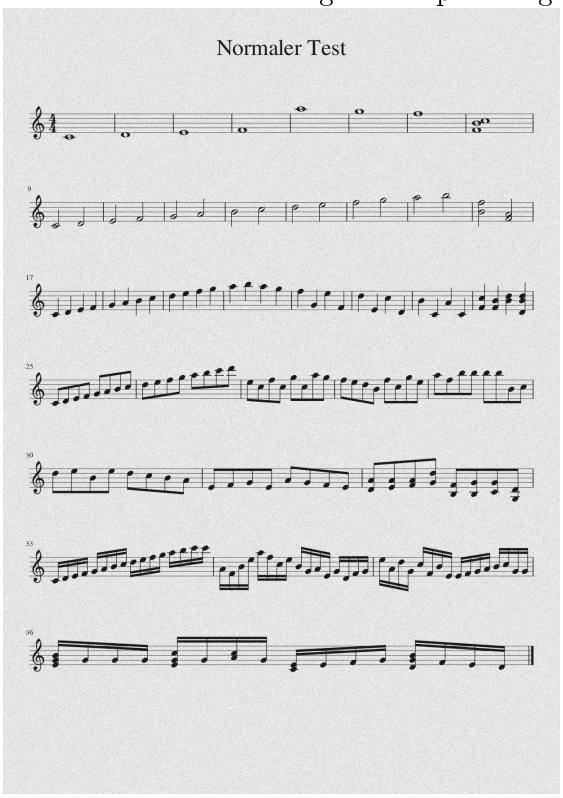
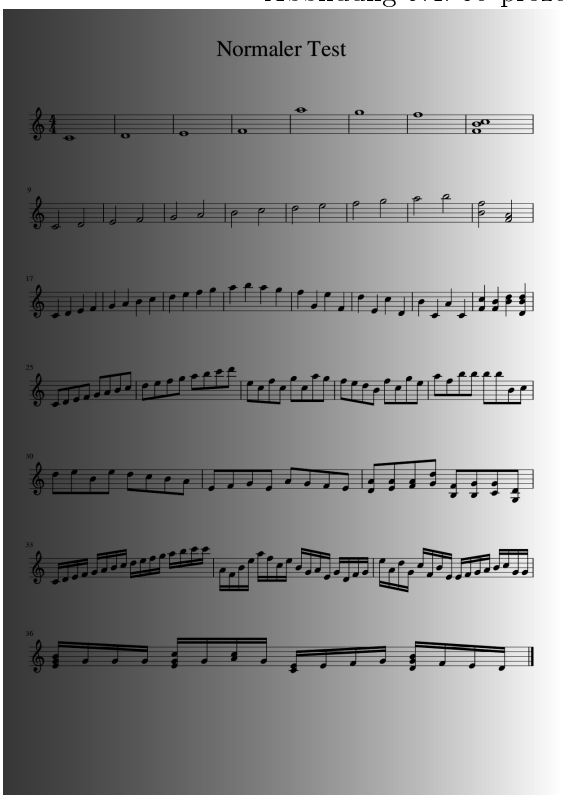


Abbildung 6.4: 50-prozentiger Grauwertverlauf



Literaturverzeichnis

[Bit17] Hervé Bitteur. Audiveris project structure, 2017.

[Jä12] Bernd Jähne. *Digitale Bildverarbeitung und Bildgewinnung*. Springer Vieweg, 2012.

[Mat17] MathWorld. Barycentric coordinates, 2017.

[Pri15] Lutz Priese. *Computer Vision: Einführung in die Verarbeitung und Analyse digitaler Bilder*. Springer Vieweg, 2015.