Oskar-Maria-Graf-Gymnasium
Neufahrn b. Freising

Abiturjahrgang
2017

S E M I N A R A R B E I T

Rahmenthema des Wissenschaftspropädeutischen Seminars:

Numerik

Leitfach: ***Informatik***

Thema der Arbeit:

Autokalibrierendes mehrfarbiges zwei dimensionales Barcodesystem - Laufzeit- und Flächennutzungsoptimierung

Verfasser/in: **Janek Blankenburg**

Kursleiter/in:**StR Baumer**

Abgabetermin: *8. November 2015*

Bewertung	Note	Notenstufe in Worten	Punkte		Punkte
schriftliche Arbeit				x 3	
Abschlusspräsentation				x 1	
				Summe:	
				Gesamtleistung nach § 61 (7) GSO = Summe:2 (gerundet)	

Datum und Unterschrift der Kursleiterin bzw. des Kursleiters

Autor
Janek Blankenburg
Elsternweg 14a, 85375 Neufahrn
janek.blankenburg@icloud.com

Arbeit im Rahmen des Wissenschaftspropädeutischen Seminars der gymnasialen Oberstufe
Numerik - Leitfach: Informatik

Autokalibrierendes mehrfarbiges zwei dimensionales Barcodesystem

Laufzeit- und Flächennutzungsoptimierung

Stand: 21.10.2016

Betreut durch:
Herr Studienrat Sven Baumer
Oskar-Maria-Graf-Gymnasium
Keltenweg 5, 85375 Neufahrn
Tel.: 08165 95760

Inhaltsverzeichnis

1. Einleitung

Diese Arbeit verfolgt das Ziel, das Auslesen eines bereits von mir (siehe [1]) entwickelten mehrfarbigen zwei dimensionalen autokalibrierenden Barcode-Systems zu verbessern. Hierbei liegt der Fokus vor allem auf der Optimierung der Rechenzeit, dem Verhältnis von benötigter Barcode-Fläche und zu speichernder Information, sowie bei der Kosten-/Nutzenoptimierung.

Der hier vorgestellte Barcode ist in seinem räumlichen Aufbau vergleichbar mit einem QR Barcode. Die Informationen sind zweidimensional in Höhe und Breite angeordnet. Somit liegen die auszulesenden Felder nicht nur neben-, sondern auch übereinander.

„Zum besseren Verständnis, sollte man die Entstehungsgeschichte des Projektes kennen. Im Informatikunterricht [...] ergab sich die Problematik, wie man das momentan analoge System zur Verwaltung der Schulbücher digitalisieren kann. Schnell kam man zu dem Entschluss, dass hierfür eine Datenbank notwendig ist. An diesem Punkt stellte sich die Frage, ob nur die einzelnen Buchtypen mit den dazu gehörenden Daten [...] aufgeführt werden, oder ob man jedes Buch individuell aufführen sollte? Hierbei würden auch alle Daten erfasst werden, die sonst nur im Einband des einzelnen Buches angegeben sind, wie z.B. Anzahl der Vorbesitzer oder mögliche Beschädigungen. Ausgehend von der umfangreichen Vorgehensweise ergab sich das Problem der Identifikation der Bücher.“[1]

Daraus entwickelte sich die Ursprungsidee zu diesem Projekt, die bis zum Februar 2015 zu dem Jugend forscht-Projekt „Mehrfarbiger zwei-dimensionaler Barcode zum Aufrufen eines Datenbankeintrags“ entwickelt wurde. Diese, im Wettbewerb vorgestellte, Version hatte noch verschiedene Schwachpunkte, die zwischenzeitlich deutlich verbessert wurden. Die vorliegende Arbeit ist eine Überarbeitung der Langfassung [2] eines gleichnamigen Jugend forscht-Projektes mit dem 2016, durch den Gewinn des Regionalwettbewerbs München-Nord, der Landeswettbewerb Bayern erreicht werden konnte.

Eine der Grundsätze des Gesamtprojektes ist es, die Anschaffungskosten bei der Einführung, z.B. durch den Kauf von spezieller Hardware, wie Lesegeräte, so gering wie möglich zu halten. So kann der Barcode mit einer beliebigen Kamera ausgelesen werden.

Die Idee dieses Projektes ist seit Februar 2015 rechtlich geschützt.

2. Begriffserklärung

„Obwohl Barcode wörtlich übersetzt „Streifencode“ heißt, kann auch für die hier vorgestellte Art der Datenspeicherung der Begriff Barcode verwendet werden, da der Begriff Barcode mittlerweile ein stehender Begriff für jegliche Form von visuellen Speicherarten ist.“ [1][10]

3. Einführung in das Barcodesystem

3.1. Aufbau

Der Barcode besteht in dem hier vorgestellten Beispiel (siehe 4. Farbdiskussion) aus 16 Feldern, von denen acht Felder die eigentlichen Informationen speichern.
Diese sind mit Zahlen in der Abbildung 1 gekennzeichnet. Die Eckfelder haben eine doppelte Funktion. Zum einen wird hieran die Drehung des Barcodes erkannt, wobei das schwarze mit „S“ gekennzeichnete Feld sich in der Orginalausrichtung immer links oben befindet. Die übrigen mit „W“ markierten Ecken sind immer weiß. Die zweite Funktion der Felder ist die Kalibration von schwarz und weiß. Die Felder „R“, „G“ und „B“ werden zur Kalibration von Rot, Grün und Blau verwendet (siehe 3.3.3 Kalibration). Bei dem Feld „K“ handelt es sich um ein Prüffeld (siehe. 3.3.1 Vermessung des Barcodes und 7.3 Verwendung eines Prüffeldes).

S	B	P	W
R	0	1	2
G	3	4	5
W	6	7	W

Abbildung 1: Aufbau des Barcodes

3.2. Erstellung

3.2.1. Umrechnung des Eingabewertes

Zunächst wird ein Eingabewert eingelesen und in das 5er-System umgerechnet. Dies geschieht mit Hilfe der Anwendung der Systematik des Horner-Schemas (vgl. Abbildung 2). Konkret wird jeder Wert pro gültigem Feld durch fünf geteilt, wobei der vorherige Wert auf Modulo fünf gesetzt wird.

Bei dem 5er-System handelt es sich um eine Umsetzung des Binärsystem in fünf Ziffern (hier Farben).

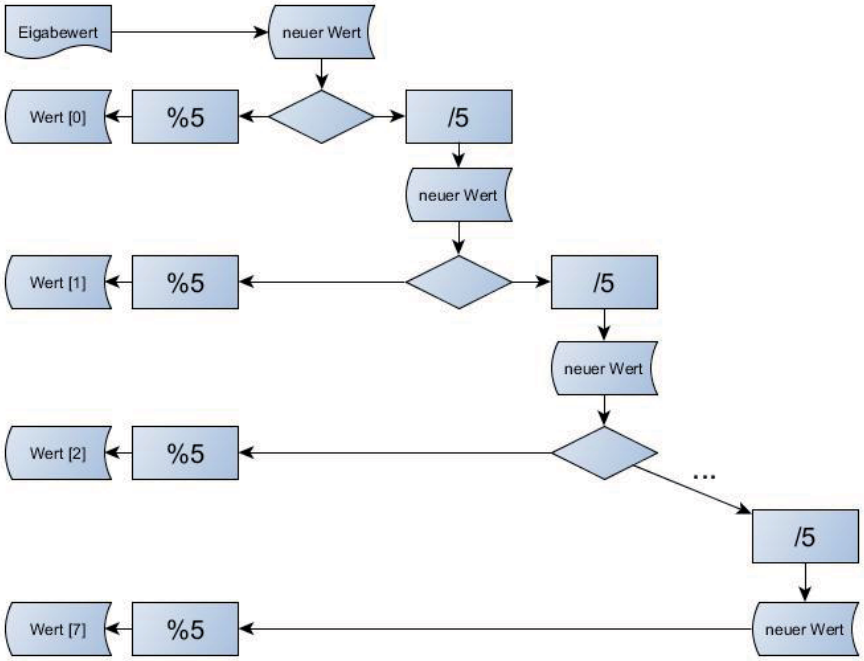


Abbildung 2: Horner-Schema[1]

3.2.2. Berechnung des Prüfwertes

Als nächstes wird der Wert des Prüffeldes berechnet. Hierfür wird nun das Array, indem die vorher berechneten Werte gespeichert wurden, von hinten ausgelesen und direkt im Dezimalsystem zusammenaddiert, wobei jede zweite Stelle von hinten nach vorne mit zwei multipliziert wird. Von dieser Summe wird der Modulo 5 Wert genommen und als

[1] angelehnt an ISO-Norm 5807 (12) beziehungsweise DIN-Norm 66001 (5)[3]

Prüfziffer festgelegt. Danach werden bereits die Farbwerte für das Prüffeld berechnet. Dies geschieht nach dem gleichen Prinzip, wie bei der Berechnung der übrigen Farbwerte. (vgl. 3.2.3 Berechnung der Farbwerte)

3.2.3. Berechnung der Farbwerte

Im nächsten Schritt werden aus den Zahlenwerten die echten Farbwerte bestimmt, wobei folgendes gilt:

Zahlwert	→	Farbe	→	Rotwert	;	Grünwert	;	Blauwert
0	→	schwarz	→	0	;	0	;	0
1	→	blau	→	0	;	0	;	255
2	→	rot	→	255	;	0	;	0
3	→	grün	→	0	;	255	;	0
4	→	weiß	→	255	;	255	;	255

Tabelle 1: Definition der Farbwerte

Die Ergebnisse werden in dem Array „ `wert[]` “ gespeichert. Die Positionierung erfolgt, wie in 3.1. gezeigt.

3.2.4. Zeichnung des Barcodes

Im finalen Schritt wird der eigentliche Barcode als Grafik erzeugt und abgespeichert, dazu wird eine Schleife gestartet, die die jeweilige Zeile komplett darstellt. Im Inneren wird eine weitere Schleife gestartet, die in jeder Zeile alle Felder darstellt, wobei der jeweilige Farbwert aus „ `wert[]` “ ausgelesen wird.

3.3. Auswertung

3.3.1. Vermessung des Barcodes

Zu Beginn des Auslesevorgangs ist die Größe eines einzelnen Feldes zu ermitteln. Es wird davon ausgegangen, dass das mittlere Pixel des Bildes Teil des Barcodes ist. Im folgenden Schritt werden von diesem Pixel, unter Anwendung des „Bresenham-Algorithmus“ [6] 16 Punkte, jeweils unter einem Winkel von $22,5°$ ($360°/16 =$) auf den Rand des Feldes, dass den mittleren Pixel beinhaltet, projiziert. Unter Verwendung dieser Schnittpunkte werden die Funktionen der einzelnen Kanten des Feldes bestimmt. Hierzu wird eine Funktion durch zwei Punkte gelegt und ein errechneter dritter Punkt mit der echten

Position verglichen. Liegen diese Punkte innerhalb des, u.a. aus Druckungenauigkeiten resultierendem, Toleranzbereiches, kann eine Funktion einer Kante bestimmt werden.

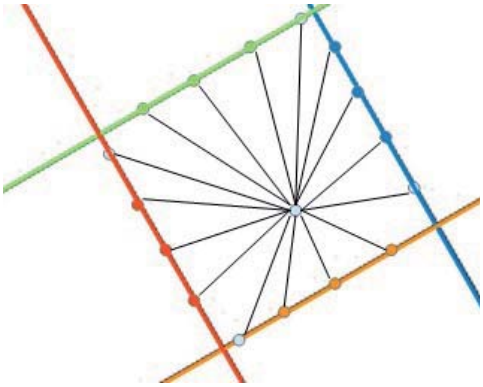


Abbildung 3: Vermessungsschema

Zum Abschluss der Vermessung des Feldes werden die Schnittpunkte der Funktionen, also gleichzeitig die Eckepunkte des Feldes errechnet. Im nächsten Schritt wird geprüft, wie viele Felder sich von dem ausgemessenem Feld nach oben und nach links anschließen. Hierzu wird der Mittelpunkt fiktiv parallel zu einer Kante um die Länge dieser verschoben. Von diesen Punkt aus wird der Vorgang zur Vermessung des Feldes wiederholt und die Längen der Diagonalen mit den Längen der Diagonalen des Ausgangsfeldes verglichen. Stimmen diese, unter Berücksichtigung eines Toleranzbereiches, miteinander übereinander, so befindet sich an der geprüften Stelle ein weiteres Feld. Durch Wiederholung dieser Methode, kann die Position des linken oberen Feldes definiert werden, von der aus, durch die ermittelten Daten, die Position jedes weiteren Feldes bestimmt werden kann.

3.3.2. Prüfung der Drehung

Im folgenden Arbeitsschritt wird ermittelt, um welchen Winkel der Barcode auf dem auszulesendem Bild gedreht ist. Dazu wird das dunkelste Feld der vier Ecken ermittelt. Bei diesem handelt es sich um das bei 0° links oben liegende schwarze Feld, aufgrund der realen Position auf dem Bild, kann der Drehwinkel das Bildes auf 90° genau bestimmt werden. Genauere Bestimmungen sind aufgrund der möglichen Abweichung bei der Feldgröße nicht notwendig.

3.3.3. Kalibration

Die entscheidende Schwäche der Vorgängerversionen war, dass die Farben nach festen Werten bestimmt wurden. Das bedeutet, es wurde geprüft, ob ein Feld, wenn es auf „Rot“ geprüft werden sollte, einen Rotanteil von mehr als 200 und einen Grün- und Blauwert von weniger als 100 besitzt. Dies führte dazu, dass ein erfolgreiches Auslesen nur bei einem sehr hochwertigen, damit teurem Druck sowie unter idealen Lichtbedingungen gewährleistet werden konnte.
Diese Problematik wurde behoben, indem die Farben in der vorliegenden Version kalibriert werden. Dies geschieht, indem das Programm, ausgehend von der schwarzen Ecke, erkennt, an welcher Stelle das blaue, rote und grüne Kalibrierungsfeld liegt (vgl. 3.1 Aufbau). Diese Felder haben immer die definierte Farbe und stehen nicht zur Informationsspeicherung zur Verfügung. Die Farben „Schwarz“ und „Weiß“ werden anhand der Eckfelder kalibriert. Nun werden die einzelnen Felder mit den zuvor definierten Farben abgeglichen und die Ergebnisse in dem Array „ `werte[]` “ festgehalten.

3.3.4. Berechnung der Zahlen nach dem Horner-Schema

Um aus den ermittelten Zahlen den Gesamtwert zu errechnen, wird das sogenannte Horner-Schema verwendet. Hierbei wird der erste Wert an der Stelle 0 des Arrays mit fünf, da mit fünf Farben gearbeitet wird, multipliziert und mit dem nächsten Wert addiert. Dies geschieht so lange, bis alle Werte verarbeitet sind.

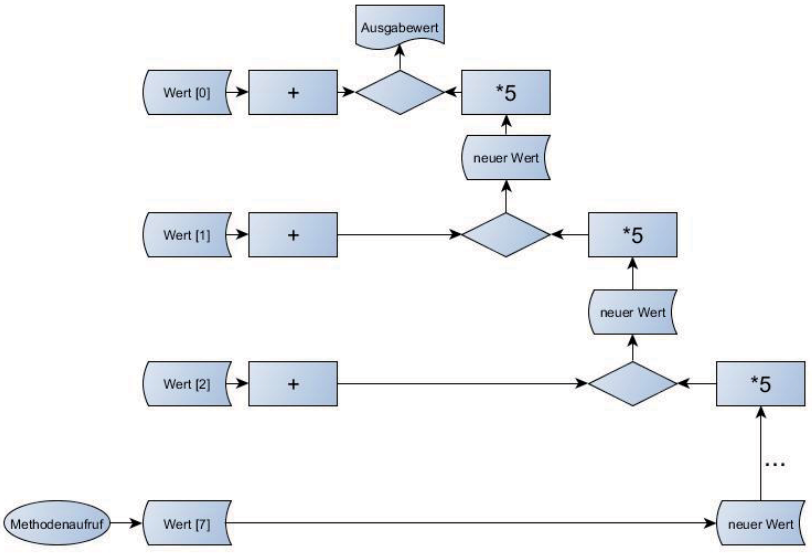


Abbildung 4: Anwendung des Horner-Schemas[2]

3.3.5. Validierung der Prüfziffer

Eine Möglichkeit der Prüfung, ob das erhaltene Ergebnis korrekt sein kann, wurde durch die Integrierung eines Prüffeldes in der vorliegenden Version erreicht (vgl. 3.1). Die Prüfung erfolgt zunächst, indem die acht gültigen Felder ausgelesen werden. Im Anschluss wird aus dem Endergebnis die theoretisch zu erzielende Prüfziffer errechnet (vgl. 3.2.2) und mit dem reellen Wert verglichen. Ein valides Endergebnis wird nur ausgegeben, wenn der ausgelesene und der berechnete Wert identisch sind.

3.3.6. Mittelpunktskorrektur

Das mittlere Pixel muss, wie oben bereits beschrieben, Bestandteil des auszulesenden Barcodes sein. Hierbei besteht allerdings die Gefahr, dass dieser Bildpunkt Teil der Umrandung der einzelnen Felder ist. Er befindet sich also in der grauen Trennfläche zwischen zwei Feldern bzw. zwischen dem Barcode und der Umgebung. Diese Situation wird gelöst, indem der Messvorgang nochmals wiederholt wird. Dabei geht man von einem um 15 Pixel nach links verschobenen Mittelpunkt aus. Um die Gefahr auszuschließen, dass der Mittelpunkt auf einer horizontalen Trennlinie liegt und folglich erneut kein gültiges Ergebnis erzielt werden kann, wird der relative Mittelpunkt nun um 15 Pixel nach oben verschoben.

4. Farbdiskussion

Es stellt sich die zentrale Frage, wie viele Farben auf welcher Fläche optimal für den größten möglichen Informationsgehalt sind. Des Weitern muss die zum Umrechnen der Werte benötigte Zeit berücksichtigt werden.
Damit über dieses Themengebiet eine Aussage getroffen werden kann, wird davon ausgegangen, dass mit insgesamt 16 Feldern gearbeitet wird. Dies ist nötig, um zunächst für die variable Anzahl der verwendeten Farben, die jeweilige Zahl der gültigen Felder berechnen zu können.

[2] angelehnt an ISO-Norm 5807 (12) beziehungsweise DIN-Norm 66001 (5)[2]

4.1. Anzahl der Farben

Abzüglich der fest belegten Felder, ergibt sich folgende Aufstellung:

Farben	n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
gültige Felder	16-3-n	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0

Tabelle 2: Anzahl der gültigen Felder, bei veränderter Anzahl von Farben

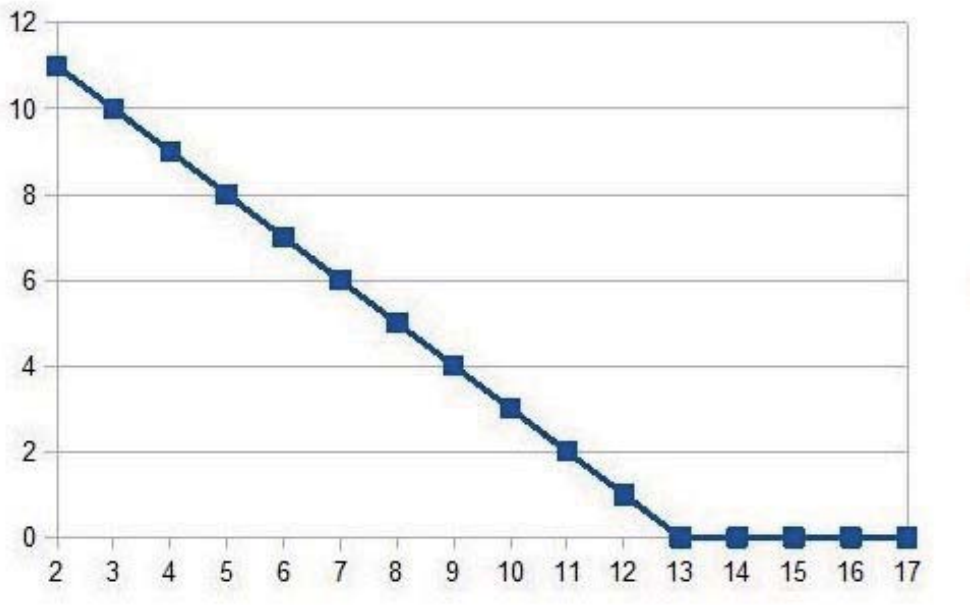


Abbildung 5: Anzahl der gültigen Felder, bei veränderter Anzahl von Farben

4.2. Anzahl der Informationen

Als nächstes wird untersucht, mit welcher Anzahl sich die meisten Informationen auf den 16 Feldern speichern lassen. Hierfür ergibt sich folgende Situation.

Farben	n	2	3	4	5	6	7	8
Informationen	n^{13-n}	2048	59 049	262 144	390 625	279 936	117 649	32 768

Farben	9	10	11	12	13	14	15	16
Informationen	6561	1000	121	12	0	0	0	0

Tabelle 3: Mögliche zu speichernde Informationen

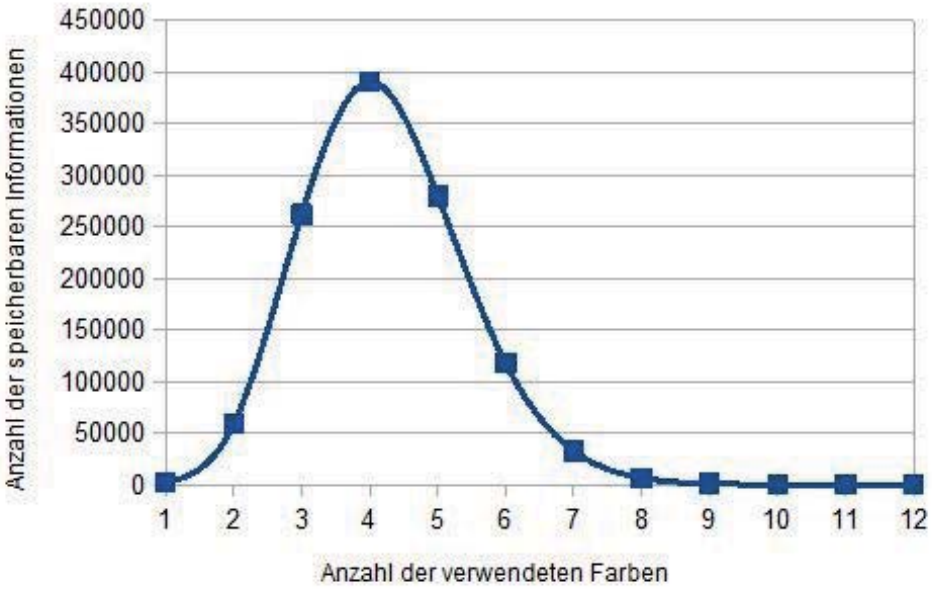


Abbildung 6: Anzahl der zu speichernden Informationen, bei veränderter Anzahl von Farben

Es zeigt sich, dass die optimale Nutzung der Fläche bei der Verwendung von fünf Farben vorliegt.

4.3. Betrachtung der Laufzeit

Eine weitere betrachtete Thematik ist die Analyse und Optimierung der benötigten Rechenzeit beim Umrechnen der Werte in das Dezimalsystem. Prinzipiell sollten Zweierpotenzen schneller das Ergebnis ermitteln, als nicht Zweierpotenzen, da bei einer Zweierpotenz nur ein „bitshift“ im Binärsystem für Multiplikationen notwendig ist. Unter Berücksichtigung dieses Aspektes wären vier Farben besser.
Zur Ermittlung des praktischen Unterschieds, wurde eine Messreihe erstellt.

	dreier-System	vierer-System	fünfer-System	sechser-System
Durchschnittszeit	139 515	137 457	135 849	134 282
Maximum	124 818 370	253 924 663	83 469 5013	84 4930 616
Minimum	99 999	100 208	99 582	99 322

Tabelle 4: Ergebnisse der Messreihe in Nanosekunden[3]

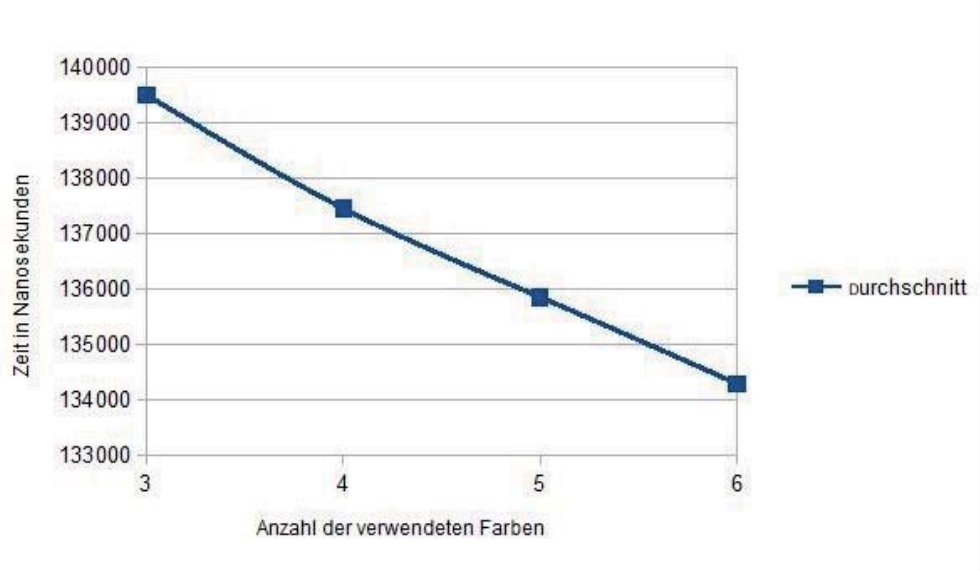


Abbildung 7: Durchschnittliche Rechenzeit bei veränderter Anzahl der Farben

Hierdurch ist zu erkennen, dass, entgegen der Vermutung, die Rechenzeit abnimmt, desto größer das Zahlensystem wird. Dies ist der Fall, da aufgrund der geringeren Anzahl gültiger Felder, die Schleife zur Berechnung der einzelnen Werte weniger häufig ausgeführt wird. Der zeitliche Unterschied ist allerdings nur sehr gering (für die Umrechnung ca. 4%, für den Gesamtprozess ca. 0,0003%). Generell ist dies also ein weiterer Grund für die Verwendung von fünf Farben.

5. Laufzeitoptimierung

„Zeit ist Geld“, diese oft zitierte Aussage kann für den vorgestellten Barcode wörtlich genommen werden. Im Rahmen einer Verwendung in einem Wirtschaftsbetrieb, kann davon ausgegangen werden, dass ein Mitarbeiter den Auslesevorgang in Gang setzt und wartet, bis dieser abgeschlossen ist. Dies führt dazu, dass die Auslesekosten des Barcodes sich aus den Betriebskosten, die im Verhältnis vernachlässigbar sind und der Arbeitszeit der Person, die den Barcode auslesen lässt, zusammensetzt. Daraus folgt, dass es, wenn es zu einer reellen Anwendung kommen soll, unerlässlich ist die Rechenzeit zu verringern. Dies wird durch Reduktion der Auslesegenauigkeit erreicht. Es stellt sich die Frage, welches Verhältnis zwischen Geschwindigkeitsgewinn und Genauigkeitsverlust erzielt werden kann.

[3] Die Messreihe entstand aus jeweils 1 000 000 Durchläufen und wurde mit einem Raspberry Pi 2 Model B: Prozessor ARM Cortex-A7, 4 Kerne, 1 GB Arbeitsspeicher, 900MHz durchgeführt.

5.1. Laufzeitoptimierung beim Auslesen

Beim Auslesen eines Barcodes gibt es zwei primäre Stellschrauben. Zum einen kann die Bildgröße bzw. die Pixelanzahl verkleinert werden, indem man z.B. nur jedes zweite Pixel ausliest. Hierdurch ergibt sich folgende Statistik.

	jedes Pixel	jedes dritte Pixel	jedes fünfte Pixel	jedes siebte Pixel
Durchschnittszeit	13 535	12 086	11 779	11 622
Maximum	15 174	12 901	12 342	12 146
Minimum	13 448	11 893	11 564	11 474

Tabelle 5: Ergebnisse der Messreihe in Millisekunden[4]

Es muss, um ein unverfälschtes Bild zu erhalten, der Prozess des Einlesen des Bildes abgezogen werden, da dieser nicht direkt beeinflusst werden kann. Messreihen haben hierfür eine Durchschnittslaufzeit von 11262 Millisekunden [4] ergeben. An dieser Stelle besteht das größte Verbesserungspotential. Hieraus ergibt sich folgende Aufstellung:

	jedes Pixel	jedes dritte Pixel	jedes fünfte Pixel	jedes siebte Pixel
Durchschnittszeit	2273	824	517	360

Tabelle 6: Ergebnisse der Messreihe in Millisekunden nach Abzug der Einlesezeit.

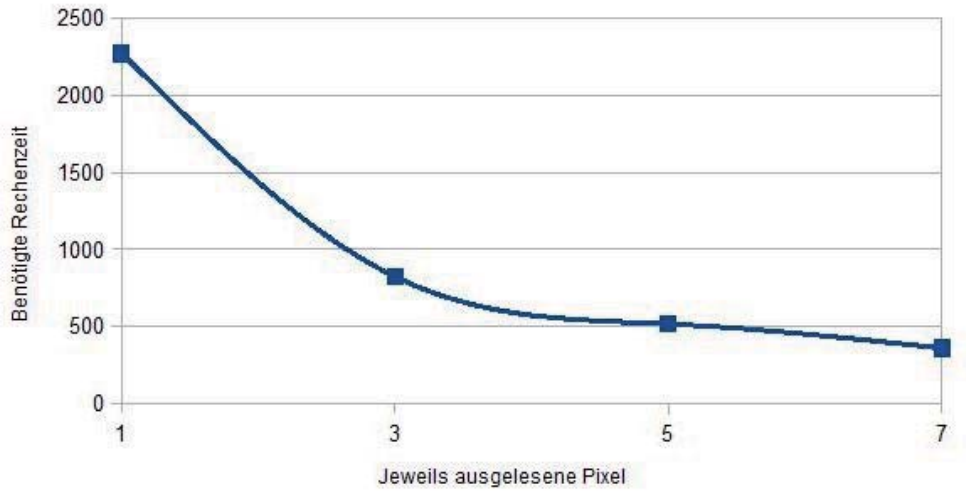


Abbildung 8: Ergebnisse der Messreihe in Millisekunden nach Abzug der Einlesezeit.

[4] Die Messreihe entstand aus jeweils 100 Durchläufen, die mit einem Raspberry Pi 2 Model B: Prozessor ARM Cortex-A7, 4 Kerne, 1 GB Arbeitsspeicher, 900MHz durchgeführt wurden. Betrachtet wurde ein Bild mit 4000Pixel · 3000Pixel=12000000Pixel.

Zum anderen können beim Messen der Größe der jeweiligen Felder nur jedes bestimmte Pixel, z.B. jedes zweite, der eingelesenen Bildpunkte geprüft werden. Logischerweise können beide Effekte kombiniert werden.

6. Kosten-/Nutzenoptimierung

Eine der häufigsten Diskussionspunkte in der Barcodethematik, ist die Gegenüberstellung von Aufwand und dem damit verbundenen technischen und wirtschaftlichen Nutzen bei einer Abweichung vom klassischen Strichbarcode.

6.1. Kostenbetrachtung

Da in dem vorgestellten System mit Farben gearbeitet wird, ist logischerweise ein Farbdruck nötig, der teurer ist, als ein schwarz-weiß Druck. Die Umgebung, in die der Barcode gedruckt wird, ist meistens bereits in Farbe gedruckt. Dies bedeutet, dass die Mehrkosten zu vernachlässigen sind, da sie sich nur auf die Druckkosten belaufen.

6.2. Flächenbetrachtung

Mit den fünf verwendeten Farben und acht gültigen Feldern lassen sich $5^8 = 390\ 625$ unterschiedliche Informationen darstellen. Der höchst mögliche Ausgabewert ist 390 624, da auch der Wert 0 definiert ist. Würde dagegen nur mit zwei Farben gearbeitet werden, könnte man, bei acht gültigen Feldern, nur 256 Ergebnisse, also ca. 0,006%, ermitteln. Hierbei muss festgehalten werden, dass man bei der Verwendung von nur zwei Farben nicht acht, sondern elf gültige Felder bei gleicher Fläche hätte. Bei dieser Umsetzung wären keine Kalibrationsfelder für die Farben rot, blau und grün nötig. Folglich könnten in einem vergleichbarem schwarz-weiß System eigentlich 2048 Informationen gespeichert werden. Diese stellen nur ca. 0,524% der mit fünf Farben möglichen Ergebnisse da.

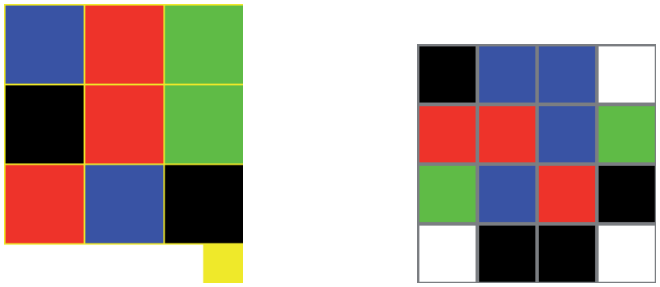
6.2.1. Flächenbedarf

Bei dieser Thematik ist logischerweise die Frage entscheidend, wie viel reale Fläche für einen farbigen Barcode benötigt wird?
Im Klartext: Wie groß muss ein Barcode gedruckt werden, damit er einlesbar ist?
Hier gibt es zwei Faktoren, die entscheidend sind. Ein Barcode kann so klein gedruckt werden, wie die Druckqualität klar differenzierte Felder zulässt.
Ist dies der Fall, kann ein Barcode mit einer handelsüblichen Digitalkamera oder einem

Smartphone ausgelesen werden, da der gesamte Markt eine höhere Auflösung, als ein handelsüblicher Drucker, zulässt.

7. Besonderheiten des Systems

Zusammenfassend kann gesagt werden, dass die 2014er Version ein reiner Prototyp war. Im Mittelpunkt stand die Entwicklung von theoretischen und praktischen Grundlagen mehrfarbiger Barcodesysteme. Der aktuelle Entwicklungsstand ist deutlich praxistauglicher und als standardisierte Routine verwendbar.



Version Dezember 2014 $\Longrightarrow$ Version Januar 2016

7.1. Autokalibration

Die wohl größte Stärke dieses Systems ist die Einführung der automatischen Farbkalibration im Januar 2016. Dies erhöht die Fehlertoleranz deutlich. Einzelne Farben müssen nicht mehr exakt abgebildet werden, sondern nur einheitlich ohne Verläufe in Helligkeit oder Farbwerten. Eine gesamte Farbe darf in ihrer Farbblance beliebig verschoben sein. Das bedeutet, ein Barcode ist problemlos auslesbar, wenn anstatt aller roten Felder einheitliche gelbe Felder vorhanden wären. Dies führt dazu, dass an die Druckqualität der Barcodes deutlich geringere Ansprüche gestellt werden, als an die Vorgängerversion. Konkret können die Barcodes nun auf Normalpapier mit einem handelsüblichen Laserdrucker bei 300 dpi gedruckt werden. Das Ziel, einen farbigen Barcode ohne bedeutenden finanziellen Mehraufwand zur Produktkennzeichnung verwenden zu können, ist damit erreicht. Ein weiterer Punkt ist, dass durch die integrierte Kalibration die Auslesegenauigkeit deutlich gesteigert werden konnte, da äußere Einflüsse, wie z.B. die Lichtverhältnisse weitestgehend vernachlässigbar sind.

7.2. Registration der Drehung

Weiterhin wurde die Funktion optimiert, die die Drehung des Barcodes erkennt. Diese arbeitet nun deutlich exakter als in der Vorgängerversion. Ermöglicht wird , dies da zwei Felder ausschließlich für die Erkennung der Rotation integriert wurden.

7.3. Verwendung eines Prüffeldes

In der aktuellen Version ist ein Prüffeld integriert. Dadurch wurde ein Kriterium entwickelt, dass die Korrektheit des ausgelesenen Wertes überprüft. Diese automatische Validierung vermeidet Fehler, die sonst nicht auffallen würden, aber bei der praktischen Anwendung zu großen Problemen führen könnten. Dies ist der Fall, wenn beispielsweise ein Feld des Barcodes, z.B. durch Verunreinigung falsch ausgelesen wird und dadurch eine falsche Information zum falsche Barcode geliefert wird. Diese Abweichung kann einem Anwender im Routinebetrieb nicht auffallen! Durch das Prüffeld wird dieser auf einen derartigen Fehler aufmerksam gemacht.

8. Implementierung und Dokumentation

Das vorgestellte Verfahren wurde vollständig in „Java8“ [4] implementiert. Für das Einlesen des Bildes wurde auf eine Bibliothek aus „Processing 3“ [7] zurückgegriffen. Es besteht eine generelle Trennung zwischen dem hier vorgestellten System zum erstellen/auslesen derBarcodeklassen und den Grafikklassen. Diese wurden entwickelt, um das ganze System praktisch nutzbar zu machen. Zur Erfüllung dieser selbstgestellten Aufgabe wurden verschiedene Tools implementiert, so z.B. eine Methode, die aus einem Pfad das Bild mit der höchsten Kennziffer (also das neueste) zurück gibt. Eine weitere Funktion ist es, generierte Barcodes direkt auszudrucken.

Klasse	**Package**	**Zeilen**	**Speicherung**
Draw	default	70	Anhang
FarbCode	default	326	Anhang
Auslesen	GUI	112	CD
Beispielanwendung	GUI	526	CD
Buchinfo	GUI	218	CD
Einstellungen	GUI	116	CD
Erstellen	GUI	105	CD
Standart_Beispielanwendung	GUI	350	CD
Show_Error	GUI	30	CD
Show_Text	GUI	30	CD
Prüfer	Tests	51	CD
UmrechnenZeitTest	Tests	117	CD
Erklärung	Visualisierung.erklärung	13	CD
Erklärung_Durchnummeriert	Visualisierung.erklärung	22	CD
Erklärung_Kontur	Visualisierung.erklärung	22	CD
Erklärung_Systemwert	Visualisierung.erklärung	62	CD
Erklärung_ausführen	Visualisierung.erklärung	15	CD
Hilfsklasse	help	68	Anhang
Hilfsklasse_Barcode	help	62	Anhang
Hilfsklasse_Bresenham	help	92	Anhang

Tabelle 7: Tabellarische Darstellung der implementierten Klasse

Bei der Implementierung wurde zu einem kleinen Anteil auf existierende Algorithmen zurückgegriffen, die teilweise exakt auf den jeweiligen Einsatz angepasst wurden, so dass z.B. der Breseham-Algorithmus für gerade Linien so verallgemeinert wurde, dass lineare Funktionen dargestellt werden können.

9. Literaturverzeichnis

[1] Corvin Meyer, Janek Blankenburg: Langfassung des Jugend-Forscht-Projektes: Mehrfarbiger 2-Dimensionaler Barcode zum Aufrufen eines Datenbankeintrags. Neufahrn b. Freising, Februar 2015.

[2]Janek Blankenburg: Langfassung des Jugend-Forscht-Projektes: Autokalibrierendes mehrfarbiges zwei dimensionales Barcodesystem - Laufzeit- und Flächennutzungsoptimierung. Neufahrn b. Freising März 2016.

[3]ISO: ISO/IEC 18004 Information technology — Automatic identification and data capture techniques — Bar code symbology — QR Code. Genf (Schweiz) 15.06.2000.

[4]Java
Online: https://www.java.com/de/
zuletzt aufgerufen am 05.03.16; 14:36 Uhr.

[5]Jugend forscht
Online: http://www.jugend-forscht.de/teilnahme/ablauf/schriftliche-arbeit.html
zuletzt aufgerufen am 18.07.16; 09:29 Uhr.

[6] Sebastian Kurfürst, Institut für Software-und Multimediatechnik TU Dresden: Rastergrafikalgorithmen. Juli 2008.

[7]Processing
Online: https://processing.org/
zuletzt aufgerufen am 05.03.16; 15:21 Uhr.

[8] Hochschule Ravensburg-Weingarten: DIN 66001 - FB Technologie und Management. Dezember 2009.

[9]Universität Stuttgart
Online: http://www.uni-stuttgart.de/ilg/files/lehre/infos_zum_studium/hausarbeiten/Eigenstaendigkeitserklaerung.pdf
zuletzt aufgerufen am 29.09.16; 08:12 Uhr.

[10] Wikipedia
Online: http://de.wikipedia.org/wiki/2D-Code
zuletzt aufgerufen am 04.01.15; 9:46 Uhr.

Sämtliche Abbildungen und Darstellung wurden eigenständig erstellt.

A. Anhang

A.1. Optimierung der Auslesemethode

Zur Vermessung eines einzelnen Feldes eines Barcodes bieten sich augenscheinlich einfachere Methoden, als die momentan implementierte. Zwei Methoden, die im folgenden vorgestellt werden, wurden vollständig implementiert und später wieder verworfen. In Testreihen wurde festgestellt, dass sich bei beiden Verfahren deutliche Schwächen bieten:

Variante 1:

Vom mittleren Pixel aus wird in vertikaler und horizontaler Richtung gemessen, wie viele Pixel auf dieser Achse liegen, die ununterbrochen die gleichen Farbwerte haben. Diese Werte werden jeweils ihrer Richtung nach summiert. Damit werden die Variablen „`einfachebreite`“ und „`einfachehoehe`“ definiert.
Zur weiteren Berechnung wird als nächstes die Position des Feldes den linken oberen Eckpunkt des Barcodes, in der Ausrichtung des Bildes bestimmt. Dazu wird gemessen, wie viele Felder links neben dem Feld mit dem mittleren Pixel des Bildes liegen. Es wird ein Arbeitspixel definiert, der um die „`einfachebreite`“ vom Mittelpunkt nach links versetzt liegt. Von diesem Pixel aus wird erneut in alle Richtungen die Anzahl der Pixel der gleichen Farbwerte bestimmt. Die Breite und Höhe dieses Farbbereiches wird in den Variablen „`arbeitsbreite`“ und „`arbeitshoehe`“ gespeichert.
Die ermittelten Werte werden mit den dementsprechenden Variablen des Referenzfeldes verglichen. Stimmen die Referenzergebnisse bei einer wählbaren Abweichung mit den Variablen überein, handelt es sich um ein gültiges Feld des Barcodes. Der Vorgang wird so oft wiederholt, bis der Vergleich der Variablen nicht übereinstimmt, welches spätestens nach der vierten Wiederholung der Fall ist.
Der Prozess wird im Anschluss vertikal nach oben versetzt wiederholt. So kann die Position des Feldes in der linken oberen Ecke festgestellt und in der Variablen „`Quadrat0wert`“ gesichert werden.

Der offensichtliche Nachteil dieser Methode ist, dass schon bei leicht gedrehten Barcodes Fehler zu erwarten sind, die das Auslesen unmöglich machen.

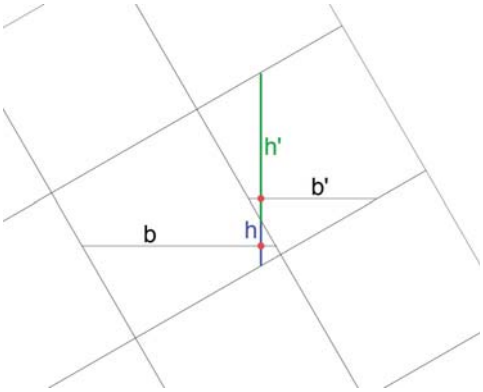


Abbildung 9: Systematik: einfache Methode

In dem Beispiel müsste $b \approx b'$ und $h \approx h'$ gelten, aber $h \ll h'$ ist der Fall. Das Feld über dem mittleren Pixel würde also nicht als gültiges Feld erkannt werden und der Barcode könnte nicht ausgelesen werden.

Variante 2:

Um dieses Problem zu lösen, war der zweite verwendete Ansatz, die Koordinaten der Eckpunkte eines einzelnen Feldes zu erkennen und zu prüfen, ob innerhalb dieser eine einheitliche Farbe vorhanden ist.

Zur Berechnung der Koordinaten werden, ausgehend vom Mittelpunkt, fünf Punkt jeweils im Abstand von 10°, unter Anwendung des „Bresenham-Algorithmus“ [6], auf den Rand des Feldes projiziert (vgl. Abbildung 10: Punkt 1). Drei der fünf Punkte liegen auf der gleichen Kante des Feldes. Man kann folglich die Koordinaten zu einer linearen Funktion vervollständigen, wobei der dritte Punkt fünf Pixel von dem theoretischen Punkt, der aus der Funktion durch die ersten beiden Punkte berechnet wurde, abweichen darf. Diese Toleranz ist unter anderem aufgrund von Druckungenauigkeiten bei der Erstellung der Barcodeetiketten notwendig. (vgl. Abbildung 10: Punkt 2)

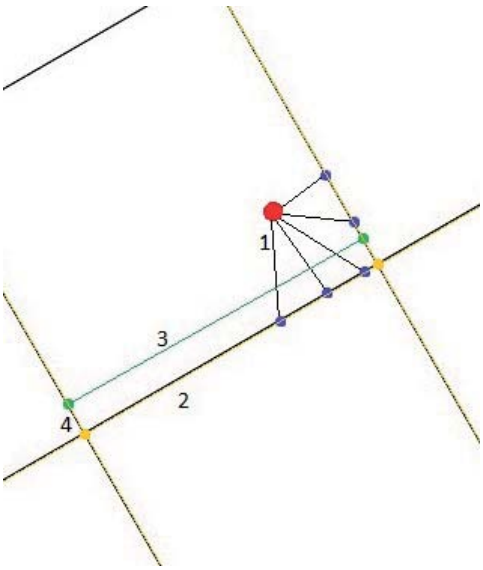


Abbildung 10: Systematik: verbesserte Methode

Im Anschluss wird die Funktion um 10 Pixel Richtung Mittelpunkt verschoben, so dass die neue Funktion parallel zur bestehenden liegt. (vgl. Abbildung 10: Punkt 3) Hieraus bilden sich wiederum zwei Schnittpunkte mit den Kanten, die im 90° Winkel zu der Ausgangsfunktion liegen. Durch diese beiden entstandenen Punkte wird ein Lot auf die Ausgangsfunktion gefällt. Die Schnittpunkte bilden zwei der Eckpunkte. (vgl. Abbildung 10: Punkt 4) Aus diesen zwei Ecken und dem Wissen um die Richtung zum Mittelpunkt, der innerhalb dieses Feldes liegt, können nun alle vier Ecken des Quadrates bestimmt werden. Damit ist die Ausrichtung, sowie die Größe eines Referenzfeldes bekannt.

Diese Methode kann nicht verwendet werden, da die Abweichung zu den Ecken, die nicht über die Schnittpunkte ermittelt werden, signifikant ist. Sie würde bei der Prüfung weiterer Felder quadratisch zunehmen, weshalb eine Erkennung der Position der linken oberen Ecke nicht immer gewährleistet werden kann.

A.2. Programmcode

A.2.1. Barcode erstellen

Mit dieser Klasse wird der Barcode ausgelesen. Hierbei handelt es sich um das Herzstück des gesamten Projektes. Zum Aufruf der Klasse muss folgender Befehl aufgerufen werden:

```
PApplet.main(new String[]{"--present", "FarbCode"});
```

```
import help.Hilfsklasse_Barcode;
import help.Hilfsklasse_Bresenham;
import processing.core.*;
\**
 * Klasse zum Auslesen von Barcodes
 * @author Janek Blankenburg
 * @version 2.5
*\
public class FarbCode extends PApplet {
    Hilfsklasse_Barcode help = new Hilfsklasse_Barcode();
    Hilfsklasse_Bresenham helpBre = new Hilfsklasse_Bresenham();
    PImage myImage;
    int tCount;
    int[][] farbwert;
    int bildbreite;
    int[] werte = new int[8];
    int[][] pruefWerte = new int[3][10];
    int einfachebreite = 0;
    int einfachehoehe = 0;
    double linkswert = 0;
    double hochwert = 0;
    int[][] kalfarben = new int[5][3];
    int pruefwert = 0;
    int mittelpunkt;
    int[] mittelpukntsfarben = new int[3];
    double[] einstell;
    int genuzt;
    String[] einstellungen;
    int masteruse = 0;
    int ap;//geschwindigkeit
    int ap_pr;
    int[] eckenabstand=new int[2];
    int[] mastereckenabstand=new int[2];
```

```
34      double[][] schnittpunkt=new double[4][2];
35      double[][] masterschnittpunkt=new double[4][2];
36      String fehler="";
37  \**
38   * Realer Konstruktor mit Mittelpunktskorrektur
39   *\
40      @Override
41      public void setup() {
42          einstellungen = help.textlesen("einstellungen.txt").split(";");
43          einstell = new double[einstellungen.length];
44          for (int i = 0; i < einstellungen.length - 1; i++) {
45              einstell[i] = Double.parseDouble(einstellungen[i]);
46          }
47          ap = (int) einstell[2];
48          ap_pr = (int) einstell[3];
49          bildeinlesen();
50          farbelesen();
51          mittelpunkt = tCount / 2 + Math.abs((myImage.height / ap \% 2) -
    1) * (bildbreite / 2);
52          master();
53          if (genuzt == 0) {
54              for (int i = 0; i < 3; i++) {
55                  switch (i) {
56                      case 0:
57                      case 1:
58                          masteruse=0;
59                          fehler="";
60                          mittelpunkt = (mittelpunkt - 15)-(15*bildbreite*i)
    ;
61                          master();
62                          if (genuzt==1){
63                              i=3;
64                          }
65                          break;
66                      case 2:
67                          if (fehler.equals("")){
68                          fehler="Die Mitte des Bildes scheint nicht Teil
    des Barcodes zu sein";
69                          }
70                          dispose();
71                          break;
72                  }
```

```
73              }
74          }
75          if(!fehler.equals("")){
76              help.error(fehler);
77          }
78      }
79  \**
80   * Hauptmethode, die den zentralen Auslesevorgang veranlässt.
81   *\
82      @SuppressWarnings("empty-statement")
83      private void master(){
84          double[] m=messfeld(mittelpunkt);
85          mastereckenabstand=eckenabstand.clone();
86          masterschnittpunkt=schnittpunkt.clone();
87          distancepruefen(1,m);
88          distancepruefen(2,m);
89          kalibrieren(m);
90          wertebestimmen();
91          wertumsetzung();
92      }
93  \**
94   * Validieren der Prüfziffer und Umrechnen der Zahl aus dem Fünfer- ins
       Dezimalsystem
95   *\
96      private void wertumsetzung(){
97          if (masteruse == 0) {
98              long gesamtwert = help.hornerrück(werte);
99              if (pruefwert == gesamtwert \% 5) {
100                 genuzt = 1;
101                 help.tetxtschreiben(gesamtwert + "", "Datei.txt", false);
102                 dispose();
103             } else {
104                 fehler="Die berechnete Prüfziffer entspricht nicht der
    Ausgelesenen!";
105                 dispose();
106             }
107         }
108         genuzt = 1;
109     }
110 \**
111  * Auslesen der Werte
112  *\
```

```
113     private void wertebestimmen() {
114         for (int i = 0; i < 8; i++) {
115             int pr = 0;
116             for (int ii = 0; ii < 5; ii++) {
117                 if (pruefWerte[0][i] >= kalfarben[ii][0] * (1 - einstell
    [0]) && pruefWerte[0][i] <= kalfarben[ii][0] * (1 + einstell[0]) &&
    pruefWerte[1][i] >= kalfarben[ii][1] * (1 - einstell[0]) && pruefWerte
    [1][i] <= kalfarben[ii][1] * (1 + einstell[0]) && pruefWerte[2][i] >=
    kalfarben[ii][2] * (1 - einstell[0]) && pruefWerte[2][i] <= kalfarben[
    ii][2] * (1 + einstell[0])) {
118                     werte[i] = ii;
119                     ii=5;
120                     pr = 1;
121                 }
122             }
123             if (pr == 0) {
124                 fehler="Der Barcode konnte nicht ausgelesen werden, wiel
    min. ein Feld keine identifizierbare Farbe besitzt.";
125                 masteruse = 1;
126             }
127         }
128     }
129 \**
130  * Prüfung der Drehung und Messung der Farbe
131  * @param m Steigung der Funktionen der Kanten
132  *\
133     void kalibrieren(double[] m) {
134         hochwert=1;
135         linkswert=1;
136         int Quadrat0wert = mittelpunkt;
137         for (int i = 0; i < hochwert; hochwert--) {
138             Quadrat0wert=mittelpunktVerschieben(Math.max(m[0],m[1]),1,
    Quadrat0wert);
139         }
140         for (int i = 0; i < linkswert; linkswert--) {
141             Quadrat0wert=mittelpunktVerschieben(Math.min(m[0],m[1]),1,
    Quadrat0wert);
142         }
143         int rechtsverschiebung=mittelpunktVerschieben(0,-1,0);
144         int tiefverschiebung=mittelpunktVerschieben(Integer.MAX_VALUE
    ,-1,0);
145         int pos = 0;
```

```
146         int[] poslist={Quadrat0wert,Quadrat0wert + rechtsverschiebung * 3,
    Quadrat0wert + rechtsverschiebung * 3 + tiefverschiebung * 3,
    Quadrat0wert + tiefverschiebung * 3};
147         int cor = -1;//drehung
148         for(int i=0;i<4;i++){
149             if (farbwert[0][poslist[i]] < 100 && farbwert[1][Quadrat0wert]
    < 100 && farbwert[2][Quadrat0wert] < 100) {
150                 pos = poslist[i];
151                 cor = i;
152             }
153         }
154         if (cor==-1){
155            fehler="Der Barcode konnte nicht identifiziert werden!";
156             dispose();
157         }
158         int Bx = (int) Math.round(0.333 * Math.pow(cor, 3) - Math.pow(cor,
    2) - 0.333 * cor + 1);// 0,333x3 - x2 - 0,333x + 1
159         int By = (int) Math.round(-0.333 * Math.pow(cor, 3) + 2 * Math.pow
    (cor, 2) - 2.667 * cor);//=Rx - 0,333x3 + 2x2 - 2,667x
160         int Ry = (int) Math.round(-0.333 * Math.pow(cor, 3) + Math.pow(cor
    , 2) + 0.333 * cor - 1);
161         int posi[] = {pos, pos + rechtsverschiebung * Bx -
    tiefverschiebung * By, pos + rechtsverschiebung * By - tiefverschiebung
     * Ry, pos + rechtsverschiebung * By * 2 - tiefverschiebung * Ry * 2,
    pos + rechtsverschiebung * By * 3 - tiefverschiebung * Ry * 3};
162         for (int i = 0; i < 5; i++) {
163             for (int ii = 0; ii < 3; ii++) {
164                 kalfarben[i][ii] = farbwert[ii][posi[i]];
165             }
166         }
167         for (int i = 0; i < 5; i++) {
168             if (farbwert[0][pos + rechtsverschiebung * Bx * 2 -
    tiefverschiebung * By * 2] > kalfarben[i][0] * (1 - einstell[0]) &&
    farbwert[0][pos + rechtsverschiebung * Bx * 2 - tiefverschiebung * By *
     2] < kalfarben[i][0] * (1 + einstell[0]) && farbwert[1][pos +
    rechtsverschiebung * Bx * 2 - tiefverschiebung * By * 2] > kalfarben[i
    ][1] * (1 - einstell[0]) && farbwert[1][pos + rechtsverschiebung * Bx *
     2 - tiefverschiebung * By * 2] < kalfarben[i][1]* (1 + einstell[0]) &&
     farbwert[2][pos + rechtsverschiebung * Bx * 2 - tiefverschiebung * By
    * 2] > kalfarben[0][2]  * (1 - einstell[0]) && farbwert[2][pos +
    rechtsverschiebung * Bx * 2 - tiefverschiebung * By * 2] < kalfarben[i
    ][2] * (1 + einstell[0])) {
```

```
169                 pruefwert = i;
170                 i=5;
171             }
172         }
173         for (int i = 0; i < 8; i++) {
174             int[] breitlist = {(i \% 3) + 1, 2 - (i / 3), 2 - (i \% 3), (i
    / 3) + 1};
175             int[] hoehelist = {(i / 3) + 1, (i \% 3) + 1, 2 - (i / 3), 2 -
    (i \% 3)};
176             for (int ii = 0; ii < 3; ii++) {
177                 pruefWerte[ii][i] = farbwert[ii][Quadrat0wert +
    rechtsverschiebung * breitlist[cor] + tiefverschiebung * hoehelist[cor
    ]];
178             }
179         }
180     }
181 \**
182  * Prüfung der Distance eines Feldes
183  * @param richtung Zu prüfende Dimension (1:hoch, 2:links)
184  * @param m Steigung der Funktionen der Kanten
185  *\
186     void distancepruefen(int richtung,double[] m) {
187         int mittelpunktNeu = mittelpunkt;
188         double im=(richtung==1)?Math.max(m[0],m[1]):Math.min(m[0],m[1]);
189         for (int i = 0; i < 4; i++) {
190             mittelpunktNeu=mittelpunktVerschieben(im,1,mittelpunktNeu);
191             double[] abbruch=messfeld(mittelpunktNeu);
192             if(eckenabstand[0]*0.8<mastereckenabstand[0]&&eckenabstand[1]*
    0.8<mastereckenabstand[1]&&eckenabstand[0]*1.2>mastereckenabstand[0]&&
    eckenabstand[1]*1.2>mastereckenabstand[1]&&(!(abbruch[0]==Integer.
    MAX_VALUE&&abbruch[1]==Integer.MAX_VALUE))){
193                 if(richtung==1){
194                     hochwert++;
195                 }else{
196                     linkswert++;
197                 }
198             }
199         }
200     }
201     \**
202  * Verschiebung des Arbeitspunkte in das nächste Feld
203  * @param richtung Zu prüfende Dimension (1:links oben, -1:rechts unten)
```

```
204  * @param m Steigung der Funktionen der Kanten
205  * @param mittelpunktNeu aktueller Arbeitspunkt
206  * @return (xn+yn*bildbreite) aktueller Arbeitspunkt
207  *\
208      int mittelpunktVerschieben(double m,int richtung,int mittelpunktNeu) {
209          if(m==Integer.MAX_VALUE){
210              int x=(mittelpunktNeu\%bildbreite);
211              int y= (int) ((mittelpunktNeu/bildbreite)-richtung*(Math.sqrt(
     Math.pow(schnittpunkt[0][1]-schnittpunkt[1][1],2)+Math.pow(schnittpunkt
     [0][0]-schnittpunkt[1][0],2))) );
212              return (x+y*bildbreite);
213          }else{
214          int t=(int) (mittelpunktNeu/bildbreite-m*Math.round(mittelpunktNeu
     \%bildbreite));
215          int dp=0;
216          dp=(int) (Math.sqrt(Math.pow(schnittpunkt[0][1]-schnittpunkt
     [1][1],2)+Math.pow(schnittpunkt[0][0]-schnittpunkt[1][0],2)));
217          int xn=(int) Math.sqrt((dp*dp)/(1+m*m));
218          xn=Math.abs(xn-richtung*(mittelpunktNeu\%bildbreite));
219          int yn=(int) (m*xn+t);
220          return (xn+yn*bildbreite);}
221      }
222      \**
223       * Abmessung eines einzelnen Feldes
224       * @param mittelpunktNeu aktueller Arbeitspunkt
225       * @return (m) Steigung der Funktionen der Kanten
226       *\
227
228       private double[] messfeld(int mittelpunktNeu){
229           int[][] bearbeitung=new int[18][2];
230           int[] prüffarbwerte=new int[3];
231           prüffarbwerte[0]=farbwert[0][mittelpunktNeu];
232           prüffarbwerte[1]=farbwert[1][mittelpunktNeu];
233           prüffarbwerte[2]=farbwert[2][mittelpunktNeu];
234          for(int i=0;i<18;i++){
235              bearbeitung[i]=helpBre.zubreseham(Math.tan((360-10*i*2)*((2*
     Math.PI)/360)),mittelpunktNeu,bildbreite,prüffarbwerte,farbwert,tCount)
     ;
236              if (i<9){
237                  int temp=bearbeitung[i][0];
238                  bearbeitung[i][0]=bearbeitung[i][2];
239                  bearbeitung[i][2]=temp;
```

```
240                 temp=bearbeitung[i][1];
241                 bearbeitung[i][1]=bearbeitung[i][3];
242                 bearbeitung[i][3]=temp;
243             }
244         }
245         int n=0;
246         double[] m=new double[4];
247         double[] t=new double[4];
248         double[] p=new double[2];//theoretischer Punkt
249         int[] p2=new int[4];
250         for(int i=0;n<4;i++){
251             int nullmode=0;
252             m[n]=0;
253             m[n]=((bearbeitung[(i+1)\%16][0]-bearbeitung[(i\%16)][0])==0)?
    nullmode=1:(bearbeitung[(i+1)\%16][1]-bearbeitung[(i\%16)][1])/(
    bearbeitung[(i+1)}\%16][0]-bearbeitung[(i\%16)][0]);
254             t[n]=bearbeitung[(i\%16)][1]-m[n]*bearbeitung[(i\%16)][0];
255             p=new double[2];
256             if(m[n]<=1){
257             p[0]=bearbeitung[(i+2)\%16][0];
258             p[1]= m[n]*bearbeitung[(i+2)\%16][0]+t[n];}
259             else{
260                 p[1]=bearbeitung[(i+2)\%16][1];
261                 p[0]=(p[1]-t[n])/m[n];
262             }
263             m[n]=(nullmode==1)?Integer.MAX_VALUE:m[n];
264             if (n>0){
265                 if(nullmode==1 && m[n-1]!=Integer.MAX_VALUE&&(!((m[n
    -1]>1||m[n-1]<-1) && (m[n]>1||m[n]<-1)))){
266                     p2[n]=(i\%16);
267                     i=i+2;
268                     n++;}else{
269                 if((m[n]-2>m[n-1]||m[n]+2<m[n-1])&&(!((m[n-1]>1||m[n
    -1]<-1) && (m[n]>1||m[n]<-1)))&& Math.abs(p[1]-bearbeitung[(i+2)
    \%16][1])<5){
270                     p2[n]=(i\%16);
271                     n++;
272                 }}
273             } else{
274                 if(nullmode==1 && Math.abs(p[0]-bearbeitung[(i+2)\%16][0])
    <5){
275                     p2[n]=(i\%16);
```

```
276                     n++;}else{
277                 if(Math.abs(p[1]-bearbeitung[(i+2)\%16][1])<5){
278                 p2[n]=(i\%16);
279                 n++;}}
280             }
281             if(i>16){
282                 m[0]=Integer.MAX_VALUE;
283                 m[1]=Integer.MAX_VALUE;
284                 return(m);
285             }
286         }
287         for(int f=0;f<4;f++){
288             if(m[f]==Integer.MAX_VALUE){
289                 schnittpunkt[f][0]=bearbeitung[p2[f]][0];
290                 schnittpunkt[f][1]=m[(f+1)\%4]*schnittpunkt[f][0]+t[(f+1)
\%4];
291             }
292             else{
293                 if(m[(f+1)\%4]==Integer.MAX_VALUE){
294                 schnittpunkt[f][0]=bearbeitung[p2[(f+1)\%4]][0];
295                 schnittpunkt[f][1]=m[f]*schnittpunkt[f][0]+t[f];
296             }else{
297             schnittpunkt[f][0]=Math.round((t[(f+1)\%4]-t[f])/(m[f]-m[(f+1)
\%4]));
298             schnittpunkt[f][1]=m[f]*schnittpunkt[f][0]+t[f];
299         }
300             }
301         }
302         eckenabstand[0]=(int) Math.sqrt(Math.pow(schnittpunkt[0][0]-
schnittpunkt[2][0], 2)+Math.pow(schnittpunkt[0][1]-schnittpunkt[2][1],
2));
303         eckenabstand[1]=(int) Math.sqrt(Math.pow(schnittpunkt[1][0]-
schnittpunkt[3][0], 2)+Math.pow(schnittpunkt[1][1]-schnittpunkt[3][1],
2));
304         return(m);
305       }
306       \**
307        * Farben eines einzelnen Feldes einlesen
308        *\
309
310      private void farbelesen() {
311         for (int i = 0; i < tCount; i++) {
```

```
312                 for (int ii = 0; ii < 3; ii++) {
313                     farbwert[ii][i] = help.farbe(myImage.pixels[i * ap], ii);
314                 }
315             }
316         }
317 \**
318  * Einlesen des Bildes
319  *\
320         public void bildeinlesen() {
321             myImage = loadImage(help.bild(einstellungen));
322             tCount = myImage.pixels.length / ap;
323             bildbreite = myImage.width / ap;
324             farbwert = new int[3][tCount];
325         }
326 }
```

A.2.2. Barcode erstellen

Mit dieser Klasse wird der Barcode erstellt und automatisch gedruckt. Zum Aufruf der Klasse muss folgender Befehl aufgerufen werden:

```
1 help.tetxtschreiben(eingabe.getText(), "Eingabewerte", false);
2 PApplet.main(new String[]{"--present", "Draw"});
```

```
1  import help.Hilfsklasse_Barcode;
2  import java.io.File;
3  import java.io.IOException;
4  import java.util.logging.Level;
5  import java.util.logging.Logger;
6  import processing.core.*;
7
8  public class Draw extends PApplet {
9
10     Hilfsklasse_Barcode help = new Hilfsklasse_Barcode();
11     PImage myImage;
12     int[] wertumsetzung = new int[9];
13     double[][] farbwerte = {{0, 0, 0}, {0, 0, 255}, {0, 0, 0}, {255, 255,
       255}, {255, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 255, 0}, {0, 0,
       0}, {0, 0, 0}, {0, 0, 0}, {255, 255, 255}, {0, 0, 0}, {0, 0, 0}, {255,
       255, 255}};
```

```
14      long zahlwert;
15      @Override
16      @SuppressWarnings("empty-statement")
17      public void setup() {
18          zahlwert=Long.parseLong(loadStrings("Eingabewerte.txt")[0]);
19          wertumsetzung = help.horner(zahlwert);
20          prüfwert();
21          farbwerte();
22      }
23
24      @Override
25      public void draw() {
26          size(1400, 1400);
27          background(255, 255, 255);
28          strokeWeight(3);
29          stroke(128, 128, 128);
30          int iii = 0;
31          for (int i = 0; i < 4; i++) {
32              for (int ii = 0; ii < 4; ii++) {
33                  fill((int) farbwerte[iii][0], (int) farbwerte[iii][1], (
    int) farbwerte[iii][2]);
34                  quad(ii * 50, i * 50, ii * 50, i * 50 + 50, ii * 50 + 50,
    i * 50 + 50, ii * 50 + 50, i * 50);
35                  iii++;
36              }
37          }
38          save("Barcode.png");
39                  String commandoToRun = "mspaint /pt Barcode.png";
40          File workoingDir = new File(sketchPath(""));
41          try {
42              Process p = Runtime.getRuntime().exec(commandoToRun, null,
    workoingDir);
43          } catch (IOException ex) {
44              Logger.getLogger(Draw.class.getName()).log(Level.SEVERE, null,
     ex);
45          }
46          size(0, 0);
47          dispose();
48      }
49
50      public void farbwerte() {
51          int i = 0;
```

```
52          for (int a = 5; a < 15; a++) {
53              if (a != 12 && a != 8) {
54                  wertberechnug(a, wertumsetzung[i]);
55                  i++;
56              }
57          }
58      }
59
60      public void prüfwert() {
61          wertberechnug(2,(int)zahlwert\%5);
62      }
63
64      public void wertberechnug(int a,int form){
65          farbwerte[a][0] = Math.round(0.292 * Math.pow(form, 4) - 2.25 *
    Math.pow(form, 3) + 5.208 * Math.pow(form, 2) - 3.25 * form) * 255;
66          farbwerte[a][1] = Math.round((form * 0.25) - 0.2) * 255;
67          farbwerte[a][2] = Math.abs(Math.round(0.125 * Math.pow(form, 4) -
    1.25 * Math.pow(form, 3) + 3.875 * Math.pow(form, 2) - 3.75 * form) *
    255);
68      }
69  }
```

A.2.3. Hilfsklassen

A.2.4. Hilfsklassen

Die eingebundenen, selbstgeschriebenen Hilfsklassen werden als Bibliotheken für Methoden verwendet, die entweder von unterschiedlichen Klassen benutzt werden, oder für das Projekt zwar notwendig sind, aber im engsten Sinn nicht explizit dazugehören.

```
1  package help;
2
3  import java.io.BufferedReader;
4  import java.io.File;
5  import java.io.FileReader;
6  import java.io.FileWriter;
7  import java.io.IOException;
8  import processing.core.PApplet;
9
10 public class Hilfsklasse extends PApplet{
11     public void tetxtschreiben(String text,String farbe,boolean append){
```

```
12          FileWriter writer;
13         File file;
14         file = new File(farbe+".txt");
15      try {
16        writer = new FileWriter(file ,append);
17        writer.write(text+"");
18        writer.write(System.getProperty("line.separator"));
19        writer.flush();
20        writer.close();
21     } catch (IOException e) {
22       e.printStackTrace();
23     }
24     }
25     public String textlesen(String quelle){
26     FileReader fr;
27         BufferedReader br;
28         try {
29             fr = new FileReader(quelle+".txt");
30             br = new BufferedReader(fr);
31             String zeile;
32             zeile = br.readLine();
33             fr.close();
34             return(zeile);
35         }
36         catch (IOException e){
37             System.out.println("Fehler beim Lesen der Datei " + quelle);
38             System.out.println(e.toString());
39         }
40      return null;
41     }
42
43     public String textlesenganz(String quelle){
44     FileReader fr;
45         BufferedReader br;
46         String Text="<html><body>";
47         try {
48             fr = new FileReader(quelle+".txt");
49             br = new BufferedReader(fr);
50
51            String zeile = br.readLine();
52
53     while( zeile != null )
```

```
54      {
55        Text = Text +"<br>"+ zeile;
56        zeile = br.readLine();
57      }
58      Text=Text+"</body></html>";
59              fr.close();
60              return(Text);
61          }
62          catch (IOException e){
63              System.out.println("Fehler beim Lesen der Datei " + quelle);
64              System.out.println(e.toString());
65          }
66       return null;
67      }
68  }
```

```
1  package help;
2
3  import GUI.Show_Error;
4  import java.io.File;
5  import java.util.ArrayList;
6  import java.util.regex.Pattern;
7
8  public class Hilfsklasse_Barcode extends Hilfsklasse {
9
10     public int farbe(int color, int shift) {
11         //shift: blau=2 gruen=1 rot=0
12         return ((color - 0xff000000) >> (Math.abs(shift - 2) * 8)) \% 256;
13     }
14
15     public int hornerrück(int[] werte) {
16         int gesamtwert = 0;
17         for (int ii = 7; ii > 0; ii--) {
18             gesamtwert = (gesamtwert + werte[ii]) * 5;
19         }
20         gesamtwert += werte[0];
21         return gesamtwert;
22     }
23
24     public int[] horner(long wertlong) {
25         int[] wertumsetzung = new int[9];
```

```
26          for (int ii = 0; ii < 9; ii++) {
27              wertumsetzung[ii] = (int) (wertlong \% 5);
28              wertlong = wertlong / 5;
29          }
30          return wertumsetzung;
31      }
32
33      public String bild(String[] einstellungen){
34          File f = new File((einstellungen[4]));
35          File[] fileArray = f.listFiles();
36          String[][] files=new String[fileArray.length][2];
37          for(int i=0;i<fileArray.length;i++){
38              files[i]=fileArray[i].toString().split(Pattern.quote( "." ) );
39              files[i][0]=files[i][0].replace(einstellungen[4]+"\\","");
40          }
41          ArrayList<String[]> listA = new ArrayList<String[]>();
42          for(int i=0;i<fileArray.length;i++){
43              if(files[i][1].equals("png") || files[i][1].equals("jpg")||
    files[i][1].equals("JPG")){
44                  listA.add(files[i]);
45              }
46          }
47          String quelle="";
48          int quel=-99;
49          for(int i=0;i<listA.size();i++){
50              if(Integer.parseInt(listA.get(i)[0].substring(listA.get(i)[0].
    length()-4, listA.get(i)[0].length()))>quel){
51                  quel=Integer.parseInt(listA.get(i)[0].substring(listA.get(
    i)[0].length()-4, listA.get(i)[0].length()));
52                  quelle=listA.get(i)[0]+"."+listA.get(i)[1];
53              }
54          }
55         return(einstellungen[4]+"\\"+quelle);
56      }
57
58      public void error(String text){
59          Show_Error n = new Show_Error(text);
60          tetxtschreiben("ERROR", "Datei.txt", false);
61      }
62  }
```

```
1  package help;
2
3  public class Hilfsklasse_Bresenham extends Hilfsklasse {
4          public int[] zubreseham(double m,int arbeitspunkt, int bildbreite,
       int[] mittelpukntsfarben, int[][] farbwert){
5          int x1=arbeitspunkt\%bildbreite;
6          int y1=arbeitspunkt/bildbreite;
7          int x2=x1;
8          int y2=y1;
9          int[] cortemp=new int[2];
10         int[] cor=new int[4];
11 if (m!=0){
12     m=1/m;}
13 System.out.println(m);
14 if (m>=1){
15     cortemp=Bresenham(x1,y1,m,1,1,1,arbeitspunkt,bildbreite,
       mittelpukntsfarben,farbwert);
16     cor[0]=cortemp[0];
17     cor[1]=cortemp[1];
18     cortemp=Bresenham(x2,y2,m,-1,-1,1,arbeitspunkt,bildbreite,
       mittelpukntsfarben,farbwert);
19     cor[2]=cortemp[0];
20     cor[3]=cortemp[1];
21             }
22 if (m<1 && m>0){
23     m=1/m;
24     cortemp=Bresenham(y1,x1,m,1,1,2,arbeitspunkt,bildbreite,
       mittelpukntsfarben,farbwert);
25     cor[2]=cortemp[0];
26     cor[3]=cortemp[1];
27     cortemp=Bresenham(y2,x2,m,-1,-1,2,arbeitspunkt,bildbreite,
       mittelpukntsfarben,farbwert);
28     cor[0]=cortemp[0];
29     cor[1]=cortemp[1];
30     }
31 if (m<=-1){
32     m=Math.abs(m);
33     cortemp=Bresenham(x1,y1,m,-1,1,1,arbeitspunkt,bildbreite,
       mittelpukntsfarben,farbwert);
34     cor[0]=cortemp[0];
35     cor[1]=cortemp[1];
36     cortemp=Bresenham(x2,y2,m,1,-1,1,arbeitspunkt,bildbreite,
```

```
    mittelpukntsfarben,farbwert);
37      cor[2]=cortemp[0];
38      cor[3]=cortemp[1];
39              }
40
41  if (m>-1 && m<0){
42      m=1/Math.abs(m);
43      cortemp=Bresenham(y1,x1,m,-1,1,2,arbeitspunkt,bildbreite,
    mittelpukntsfarben,farbwert);
44      cor[0]=cortemp[0];
45      cor[1]=cortemp[1];
46      cortemp=Bresenham(y2,x2,m,1,-1,2,arbeitspunkt,bildbreite,
    mittelpukntsfarben,farbwert);
47      cor[2]=cortemp[0];
48      cor[3]=cortemp[1];
49  }
50
51  if (m==0){
52      cortemp=Bresenham(x1,y1,m,1,0,1,arbeitspunkt,bildbreite,
    mittelpukntsfarben,farbwert);
53      cor[0]=cortemp[0];
54      cor[1]=cortemp[1];
55      cortemp=Bresenham(y2,x2,m,1,-1,2,arbeitspunkt,bildbreite,
    mittelpukntsfarben,farbwert);
56      cor[2]=cortemp[0];
57      cor[3]=cortemp[1];
58          }
59  return cor;
60      }
61
62  private int[] Bresenham(int schnell,int langsam,double m,int richtung1,int
     richtung2,int verteilung,int arbeitspunkt, int bildbreite,int[]
    mittelpukntsfarben, int[][] farbwert) {
63          double fehler=0;
64          int xc=arbeitspunkt\%bildbreite;
65           int yc=arbeitspunkt/bildbreite;
66      while (mittelpukntsfarben[0] + 40 >= farbwert[0][arbeitspunkt] &&
    mittelpukntsfarben[0] - 40 <= farbwert[0][arbeitspunkt] &&
    mittelpukntsfarben[1] + 40 >= farbwert[1][arbeitspunkt] &&
    mittelpukntsfarben[1] - 40 <= farbwert[1][arbeitspunkt] &&
    mittelpukntsfarben[2] + 40 >= farbwert[2][arbeitspunkt] &&
    mittelpukntsfarben[2] - 40 <= farbwert[2][arbeitspunkt]){
```

```
67          schnell=schnell+1*richtung1;
68          fehler=fehler-1;
69          if (fehler<0){
70              langsam=langsam-1*richtung2;
71              fehler=fehler+m;}
72          if (verteilung==1){
73              xc=schnell;
74              yc=langsam;}
75          else{
76              yc=schnell;
77              xc=langsam;
78          }
79          arbeitspunkt=yc*bildbreite+xc;
80      }
81      int[] cor={xc,yc};
82      return cor;
83      }
84  }
```

A.3. Eigenständigkeitserklärung

„Hiermit versichere ich, dass ich die vorliegende Seminararbeit selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z.B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.“ [9]