Felix-Klein-Gymnasium
Böttingerstraße 17
37073 Göttingen

Facharbeit im Seminarfach Informatik

Korrektur der perspektivischen Verzerrung von fotografierten Rechtecken

Verfasser: Hannah Schlüter
Fachlehrer: Herr Flemming
Abgabetermin: 14. März 2016
Ort und Datum: Göttingen, den 20. März 2016

Inhaltsverzeichnis

1 Einleitung

Mit dem Smartphone haben viele Menschen im Alltag immer eine Kamera in der Tasche. Daher ist es wesentlich bequemer schnell ein Foto zu machen, anstatt zum Kopierer oder Scanner zu gehen. Oft fotografieren wir einen Zettel, ein Plakat, ein Schild oder Seiten aus einem Buch, wenn wir uns etwas merken oder es später lesen wollen. Dafür gibt es auch einige Apps auf dem Markt, die dem Benutzer ermöglichen sein Handy anstelle eines Scanners zu verwenden, indem sie ein schiefes Foto vom gewünschten Dokument so korrigieren, dass es in seinen Proportionen wieder dem Original entspricht.

Der rechteckige Umriss des fotografierten Dokuments ist hierbei oft nur leicht verzerrt, da sich die Benutzer Mühe geben, ein möglichst gerades Foto zu schießen. Beispielsweise in der Kriminaltechnik kann es jedoch auch von Interesse sein, stark perspektivisch verzerrte Flächen zu korrigieren.

Ein Foto ist ein zweidimensionales Abbild unserer dreidimensionalen Wirklichkeit. Durch das Verschwinden einer Dimension gehen Informationen verloren, sodass man die Projektion nicht ohne Weiteres umkehren kann. Daher werden zusätzliche Informationen über die geometrischen Eigenschaften der fotografierten Fläche genutzt. Dazu eignen sich zum Beispiel bekannte verzerrte Rechtecke, Kreise oder die Lage von Fluchtpunkten.[1]

Im Folgenden soll die Frontalansicht eines Rechtecks mithilfe eines Fotos aus einer anderen Perspektive rekonstruiert werden. Die Kamera wird in ihren Eigenschaften auf das Modell der Lochkamera beschränkt. Die Projektion vom Original zum Bild soll umgekehrt werden. Eine Java-Applikation führt dies bei verschiedenen Beispielen durch. Ziel ist es, auf diese Weise auch Schrift, die aufgrund der perspektivischen Verzerrung unleserlich geworden ist, mithilfe des Erarbeitetem wieder lesbar zu machen.

[1] vgl. JOHNSON, MICAH/FARID, HANY: Metric Measurements on a Plane from a Single Image. Aus: `http://www.mit.edu/~kimo/publications/metric/metric06.pdf`, Stand:10.02.16

2 Umkehrung der Projektion

Die Projektion vom Original zum Bild soll umgekehrt werden. Um die Projektion besser nachvollziehen zu können, wird das Modell der Lochkamera verwendet. Anschließend werden geometrische Eigenschaften des Rechtecks, dem das Viereck im Bild im Original entsprechen soll, verwendet. Ein Rechteck hat parallele, gleich lange Seiten und alle Innenwinkel sind rechte Winkel. Mit diesen Informationen werden die Koordinaten des Rechtecks in der Originalebene bestimmt und schließlich jedem Punkt im Rechteck ein Bildpunkt zugeordnet.

2.1 Projektion vom Original zum Bild am Modell der Lochkamera

Für die Kamera wird das Modell der Lochkamera (vgl. Kapitel 6.3) verwendet. Das heißt, dass die Lichtstrahlen vom Originalobjekt durch ein Loch, welches das optische Zentrum bildet, auf einen Schirm treffen. Die Größe des Bildes auf dem Schirm ist abhängig vom Abstand zwischen dem Schirm und dem optischen Zentrum. Dieser hat aufgrund des Strahlensatzes jedoch keinen Einfluss auf die Längenverhältnisse und Winkel im Bild.

Eine Punktspieglung des Bildes am optischen Zentrum bildet es so im Raum ab, dass die Lichtstrahlen, welche wie Geraden im Raum behandelt werden, ebenso wie zuvor jeweils durch den entsprechenden Bildpunkt und Originalpunkt verlaufen (vgl. Abbildung 1).

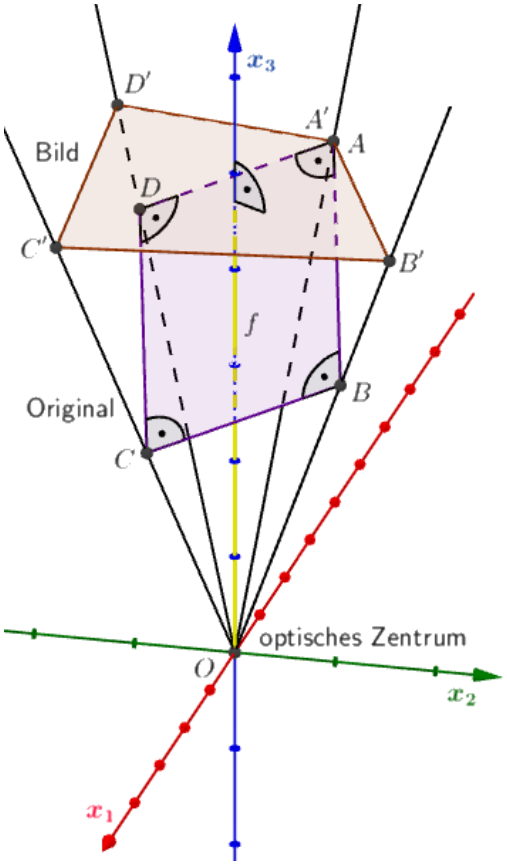


Abbildung 1: Projektion vom Original zum Bild

2.2 Koordinatensystem

Das optische Zentrum sei der Koordinatenursprung. Die erste und zweite Achse bilden eine zur Bildebene parallele Ebene. Die dritte Achse steht orthogonal zur Bildebene, sodass sie durch den Mittelpunkt des Bildes verläuft und alle Bildpunk-

te die gleiche x_3-Koordinate haben (vgl. Abbildung 1). Die x_1- und x_2-Koordinaten der Bildpunkte lassen sich folglich als Differenz der entsprechenden Pixelkoordinaten und der halben Breite beziehungsweise Länge des Bildes ausdrücken.

2.3 Wahl der Punkte und Koordinaten

Das Viereck $A'B'C'D'$ mit der Eckpunkten $A'(a_1|a_2|f)$, $B'(b_1|b_2|f)$, $C'(c_1|c_2|f)$ und $D'(d_1|d_2|f)$ in der Bildebene sei das Bild des Rechtecks $ABCD$ in der Originalebene (vgl. Abbildung 1). Daher sind die Ortsvektoren von A, B, C und D Vielfache der Ortsvektoren von A', B', C' und D', weil sie je auf der Ursprungsgeraden durch den entsprechenden Bildpunkt liegen.

Um dies darzustellen, werden die Koeffizienten λ_a, λ_b, λ_c und λ_d verwendet. Es sei $\lambda_a, \lambda_b, \lambda_c, \lambda_d \in \mathbb{R}_+$, weil die Vierecke $ABCD$ und $A'B'C'D'$ nach der Punktspiegelung am optischen Zentrum auf der selben Seite der x_1x_2-Ebene liegen. Da eine zentrische Streckung der Originalebene am optischen Zentrum nur die Größe des Originalrechtecks verändert, wird ohne Beschränkung der Allgemeinheit festgelegt, dass $\lambda_a = 1$ gelte. Die Ortsvektoren der Originalpunkte A, B, C und D seien $\vec{a}$, $\vec{b}$, $\vec{c}$ und $\vec{d}$:

$$\vec{a} = \begin{pmatrix} a_1 \\ a_2 \\ f \end{pmatrix}; \quad \vec{b} = \lambda_b \begin{pmatrix} b_1 \\ b_2 \\ f \end{pmatrix}; \quad \vec{c} = \lambda_c \begin{pmatrix} c_1 \\ c_2 \\ f \end{pmatrix}; \quad \vec{d} = \lambda_d \begin{pmatrix} d_1 \\ d_2 \\ f \end{pmatrix}. \tag{1}$$

2.4 Abstand von Bild und optischem Zentrum

Da das Viereck $ABCD$ ein echtes Rechteck ist, sind alle Innenwinkel rechte Winkel. Daher gilt:

$$\cos(\sphericalangle BAD) = 0 = \frac{\overrightarrow{AB} \cdot \overrightarrow{AD}}{|\overrightarrow{AB}| \cdot |\overrightarrow{AD}|} \Leftrightarrow \overrightarrow{AB} \cdot \overrightarrow{AD} = 0 \tag{2}$$

Die Ortsvektoren der Originalpunkte aus (1) werden eingesetzt. Anschließend wird die Gleichung nach dem Abstand f von Bild und optischem Zentrum umgeformt:

$$\begin{aligned} &\left(\lambda_b \begin{pmatrix} b_1 \\ b_2 \\ f \end{pmatrix} - \begin{pmatrix} a_1 \\ a_2 \\ f \end{pmatrix}\right) \cdot \left(\lambda_d \begin{pmatrix} d_1 \\ d_2 \\ f \end{pmatrix} - \begin{pmatrix} a_1 \\ a_2 \\ f \end{pmatrix}\right) = 0 \\ \Leftrightarrow\ &(\lambda_b b_1 - a_1)(\lambda_d d_1 - a_1) + (\lambda_b b_2 - a_2)(\lambda_d d_2 - a_2) \\ &\qquad + (\lambda_b - 1)(\lambda_d - 1) \cdot f^2 = 0 \end{aligned} \tag{3}$$

Wenn $(\lambda_b - 1)(\lambda_d - 1) \neq 0$ gilt[2], dann folgt aus (3):

$$\frac{(\lambda_b b_1 - a_1)(\lambda_d d_1 - a_1) + (\lambda_b b_2 - a_2)(\lambda_d d_2 - a_2)}{(\lambda_b - 1)(1 - \lambda_d)} = f^2 \tag{4}$$

Weil das Bild einen positiven Abstand vom optischen Zentrum haben muss, sollte f positiv sein, sodass es nur maximal eine Lösung für f gibt. Die Gleichung führt somit zu einer sinnvollen Lösung, wenn der Term auf der linken Seite positiv ist. Dann gilt:

$$f = \sqrt{\frac{(\lambda_b b_1 - a_1)(\lambda_d d_1 - a_1) + (\lambda_b b_2 - a_2)(\lambda_d d_2 - a_2)}{(\lambda_b - 1)(1 - \lambda_d)}} \tag{5}$$

Der Abstand f zwischen dem Bild und dem optischen Zentrum lässt sich nun durch Einsetzen der x_1- und x_2-Komponenten der Ortsvektoren der Punkte A, A', B, B', C, C', D und D' mit (5) berechnen. Diese Komponenten sind für die Bildpunkte schon bekannt (vgl. Kapitel 2.3). Da f die x_3-Komponente aller Bildpunkte ist, sind nun alle Komponenten der Ortsvektoren der Bildpunkte bekannt.

2.5 Parameterdarstellung der Originalebene

Im Rechteck $ABCD$ sind gegenüberliegende Seiten parallel und gleich lang, sodass $\overrightarrow{BA} = \overrightarrow{CD}$ gilt. Einsetzen der Ortsvektoren von A, B, C und D aus (1) liefert:

$$\begin{pmatrix} a_1 \\ a_2 \\ f \end{pmatrix} - \lambda_b \begin{pmatrix} b_1 \\ b_2 \\ f \end{pmatrix} = \lambda_d \begin{pmatrix} d_1 \\ d_2 \\ f \end{pmatrix} - \lambda_c \begin{pmatrix} c_1 \\ c_2 \\ f \end{pmatrix} \tag{6}$$

Es folgt ein lineares Gleichungssystem mit drei Gleichungen und drei Variablen:

$$a_1 - \lambda_b \cdot b_1 = \lambda_d \cdot d_1 - \lambda_c \cdot c_1 \tag{7}$$

$$a_2 - \lambda_b \cdot b_2 = \lambda_d \cdot d_2 - \lambda_c \cdot c_2 \tag{8}$$

$$f - \lambda_b \cdot f = \lambda_d \cdot f - \lambda_c \cdot f \tag{9}$$

Dieses wird nun nach λ_b, λ_c und λ_d aufgelöst, um die Ortsvektoren der Originalpunkte in Abhängigkeit von den Koordinaten der Bildpunkte darstellen zu können.

Wenn $d_1 \neq 0$ gilt, dann folgt aus (7):

$$\lambda_d = \frac{a_1 - \lambda_b \cdot b_1 + \lambda_c \cdot c_1}{d_1} \tag{10}$$

Weil der Abstand f zwischen dem Bild und dem optischen Zentrum nicht 0 sein darf, folgt aus (9):

$$\lambda_d = 1 - \lambda_b + \lambda_c \tag{11}$$

[2] Kapitel 2.7 befasst sich mit den Einschränkungen zur Lage der Eckpunkte des verzerrten Rechtecks.

Durch Gleichsetzen von (10) und (11) erhält man:

$$\begin{aligned} 1 - \lambda_b + \lambda_c &= \frac{a_1 - \lambda_b \cdot b_1 + \lambda_c \cdot c_1}{d_1} \\ \Leftrightarrow d_1 - \lambda_b \cdot d_1 + \lambda_c \cdot d_1 &= a_1 - \lambda_b \cdot b_1 + \lambda_c \cdot c_1 \\ \Leftrightarrow d_1 - \lambda_b \cdot d_1 - a_1 + \lambda_b \cdot b_1 &= \lambda_c \cdot c_1 - \lambda_c \cdot d_1 \end{aligned} \tag{12}$$

Wenn $c_1 - d_1 \neq 0$ gilt, dann folgt aus (12) für λ_c:

$$\lambda_c = \frac{d_1 + \lambda_b \cdot (b_1 - d_1) - a_1}{c_1 - d_1} \tag{13}$$

Einsetzten von (11) für λ_d in (8) liefert:

$$\begin{aligned} a_2 - \lambda_b \cdot b_2 &= (1 - \lambda_b + \lambda_c)d_2 - \lambda_c \cdot c_2 \\ \Leftrightarrow a_2 - \lambda_b \cdot b_2 + \lambda_b \cdot d_2 - d_2 &= \lambda_c \cdot d_2 - \lambda_c \cdot c_2 \end{aligned} \tag{14}$$

Wenn $d_2 - c_2 \neq 0$ gilt, dann folgt aus (14) für λ_c:

$$\lambda_c = \frac{a_2 + \lambda_b \cdot (d_2 - b_2) - d_2}{d_2 - c_2} \tag{15}$$

Durch Gleichsetzen von (13) und (15) erhält man:

$$\begin{aligned} &\frac{d_1 + \lambda_b \cdot (b_1 - d_1) - a_1}{c_1 - d_1} = \frac{a_2 + \lambda_b \cdot (d_2 - b_2) - d_2}{d_2 - c_2} \\ \Leftrightarrow\ &(d_2 - c_2)(d_1 + \lambda_b \cdot (b_1 - d_1) - a_1) = (c_1 - d_1)(a_2 + \lambda_b \cdot (d_2 - b_2) - d_2) \\ \Leftrightarrow\ &\lambda_b(b_1 - d_1)(d_2 - c_2) - \lambda_b(d_2 - b_2)(c_1 - d_1) \\ &\qquad = (a_2 - d_2)(c_1 - d_1) - (d_1 - a_1)(d_2 - c_2) \end{aligned} \tag{16}$$

Wenn $(b_1 - d_1)(d_2 - c_2) - (d_2 - b_2)(c_1 - d_1) \neq 0$ gilt, folgt aus (16) für λ_b:

$$\lambda_b = \frac{(a_2 - d_2)(c_1 - d_1) - (d_1 - a_1)(d_2 - c_2)}{(b_1 - d_1)(d_2 - c_2) - (d_2 - b_2)(c_1 - d_1)} \tag{17}$$

λ_b lässt sich mit (17) ausschließlich unter Verwendung der Komponenten der Ortsvektoren von A', B', C' und D' berechnen. Ebenso gilt dies auch für λ_c, wenn man (17) in (15) einsetzt und für λ_d, wenn man (17) in (15) und dann dies mit (17) in (11) einsetzt. Damit lassen sich die Ortsvektoren der Eckpunkte des Rechtecks $ABCD$ in der Originalebene mit (1) aus den Ortsvektoren der Eckpunkte des Vierecks $A'B'C'D'$ im Bild berechnen. Weil das Rechteck $ABCD$ in der Originalebene bekannt ist, lässt sich die Originalebene E mit $r, s \in \mathbb{R}$ als Parametergleichung darstellen:

$$E : \vec{x} = \vec{a} + \frac{r}{|\overrightarrow{AB}|} \cdot \overrightarrow{AB} + \frac{s}{|\overrightarrow{AD}|} \cdot \overrightarrow{AD} \tag{18}$$

Weil $ABCD$ ein Rechteck ist, sind die Richtungsvektoren von E orthogonal zueinander. Deshalb lassen sie sich als Achsen eines zweidimensionalen kartesischen

Koordinatensystem auf der Originalebene mit Koordinatenursprung A verwenden. Die Parameter wurden durch die Länge der Richtungsvektoren dividiert. Alle Punkte, deren Ortsvektor mit $0 \leq r \leq |\overrightarrow{AB}|$ und $0 \leq s \leq |\overrightarrow{AD}|$ die Parametergleichung der Originalebene (18) erfüllen, liegen im Rechteck $ABCD$. Die Punkte des Rechtecks $ABCD$ mit ganzzahligen Parametern r und s werden als Pixel im Ausgabebild verwendet.

2.6 Zuordnung der Bildpunkte zu den zugehörigen Originalpunkten

Der Originalpunkt, der zugehörige Bildpunkt und das optische Zentrum bilden eine Gerade. Der Ortsvektor eines Bildpunktes[3] $P(p_1|p_2|f)$ in der Originalebene sei $\vec{p}$. Folglich ist der Ortsvektor des zugehörigen Originalpunktes $\lambda_p \cdot \vec{p}$ mit $\lambda_p \in \mathbb{R}_+$. Es sei $\vec{u} = \frac{1}{|\overrightarrow{AB}|} \cdot \overrightarrow{AB}$ und $\vec{v} = \frac{1}{|\overrightarrow{AD}|} \cdot \overrightarrow{AD}$. Somit folgt für die Parameterdarstellung der Originalebene aus (18):

$$E : \vec{x} = \begin{pmatrix} a_1 \\ a_2 \\ f \end{pmatrix} + r \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} + s \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{19}$$

Da der Originalpunkt in dieser Ebene liegt, folgt aus (19):

$$\lambda_p \begin{pmatrix} p_1 \\ p_2 \\ f \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ f \end{pmatrix} + r \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} + s \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \tag{20}$$

Die Komponentengleichungen von (20) entsprechen dem folgenden linearen Gleichungssystem:

$$\lambda_p \cdot p_1 = a_1 + r \cdot u_1 + s \cdot v_1 \tag{21}$$

$$\lambda_p \cdot p_2 = a_2 + r \cdot u_2 + s \cdot v_2 \tag{22}$$

$$\lambda_p \cdot f = f + r \cdot u_3 + s \cdot v_3 \tag{23}$$

Das Gleichungssystem wird nach p_1 und p_2 umgeformt, um die Koordinaten des Bildpunktes P zu ermitteln.

Weil λ_p positiv ist, folgt aus (21) und (22):

$$(21) \Leftrightarrow p_1 = \frac{a_1 + r \cdot u_1 + s \cdot v_1}{\lambda_p} \tag{24}$$

$$(22) \Leftrightarrow p_2 = \frac{a_2 + r \cdot u_2 + s \cdot v_1}{\lambda_p} \tag{25}$$

[3] Alle Punkte mit der x_3-Koordinate f liegen in der Bildebene, da diese orthogonal zur x_3-Achse steht (vgl. Kapitel 2.3).

Weil f positiv sein soll, folgt aus (23):

$$\lambda_p = \frac{f + r \cdot u_3 + s \cdot v_3}{f} \tag{26}$$

Durch Einsetzen von (26) in (24) erhält man:

$$p_1 = \frac{f \cdot (a_1 + r \cdot u_1 + s \cdot v_1)}{f + r \cdot u_3 + s \cdot v_3} \tag{27}$$

Einsetzen von (26) in (25) liefert:

$$p_2 = \frac{f \cdot (a_1 + r \cdot u_2 + s \cdot v_2)}{f + r \cdot u_3 + s \cdot v_3} \tag{28}$$

Aus (27) und (28) erhält man die Koordinaten des dem Originalpunkt zugehörigen Bildpunktes:

$$P\left(\frac{f \cdot (a_1 + r \cdot u_1 + s \cdot v_1)}{f + r \cdot u_3 + s \cdot v_3} \middle| \frac{f \cdot (a_1 + r \cdot u_2 + s \cdot v_2)}{f + r \cdot u_3 + s \cdot v_3} \middle| f\right)$$

Für jedes Pixel im Ausgabebild können nun geeignete r und s gewählt werden und das zugehörige Pixel im Eingabebild bestimmt werden, sodass alle Pixel im Ausgabebild dem Eingabebild entsprechend gefärbt werden können.

2.7 Einschränkungen

In Kapitel 2.4 und 2.5 wurden beim Umformen der Gleichungen Einschränkungen zur Lage der Eckpunkte des verzerrten Rechtecks gemacht, damit weder durch 0 dividiert, noch die Wurzel aus einer negativen Zahl gezogen wird. Es soll gelten:

$$(\lambda_b - 1)(1 - \lambda_d) \neq 0 \tag{29}$$

$$\frac{(\lambda_b b_1 - a_1)(\lambda_d d_1 - a_1) + (\lambda_b b_2 - a_2)(\lambda_d d_2 - a_2)}{(\lambda_b - 1)(1 - \lambda_d)} > 0 \tag{30}$$

$$d_1 \neq 0 \tag{31}$$

$$c_1 - d_1 \neq 0 \tag{32}$$

$$d_2 - c_2 \neq 0 \tag{33}$$

$$(b_1 - d_1)(d_2 - c_2) - (d_2 - b_2)(c_1 - d_1) \neq 0 \tag{34}$$

Die Fälle, bei denen diese Einschränkungen nicht zutreffen, wurden nicht behandelt. In diesen Fällen kommt kein geeignetes Ausgabebild zustande.

3 Implementierung

Die Implementierung erfolgt in Java und nutzt das Konzept der objektorientierten Programmierung. Die Klassen heißen `Fenster`, `Korrektor`, `Ebene` und `Vektor3`.[4] Der Benutzer gibt ein Eingabebild ein und wählt ein perspektivisch verzerrtes Rechteck aus. Das Programm soll dann ein Ausgabebild mit dem korrigierten Rechteck produzieren.[5]

3.1 Klassen

3.1.1 Fenster

Die `Fenster`-Klasse (vgl. Quelltext 1) ist für die grafische Darstellung des Programms und die Verwaltung der Benutzereingaben zuständig. Sie enthält die `main`-Methode.

Für das Attribut `imgIn` wird das angegebene Eingabebild des Benutzers mittels der `importImage`-Methode eingelesen. Das Attribut `dots` enthält die Bildschirm-Koordinaten der Eckpunkte des ausgewählten verzerrten Rechtecks relativ zur linken oberen Ecke des Eingabebildes. Diese werden als Pixelkoordinaten relativ zum Mittelpunkt des Eingabebildes zusammen mit dem Eingabebild an den `korrektor` übergeben. Dann wird die `entzerren`-Methode des `korrektor` aufgerufen, welche das korrigierte Ausgabebild zurückgibt. Dieses wird im Attribut `imgOut` gespeichert und gegebenenfalls mithilfe der `scaleDinA`-Methode skaliert, um dem DIN-A-Format zu entsprechen. Anschließend wird es als Bilddatei exportiert.

Auf dem Bildschirm werden die Textfelder und Knöpfe für die Benutzereingaben sowie das Eingabe- und Ausgabebild, sobald diese vorhanden sind, im Programmfenster dargestellt. Die Benutzeroberfläche wurde mithilfe des Designeditors der NetBeans IDE entwickelt.

3.1.2 Korrektor

Ein `Korrektor`-Objekt (vgl. Quelltext 2) soll aus dem perspektivisch verzerrten Rechteck im Eingabebild `imgIn` das korrigierte Ausgabebild `imgOut` konstruieren. Wenn die `entzerren`-Methode zur Rückgabe des Ausgabebildes aufgerufen wird, wird zunächst die `findOrignalEbene`-Methode aufgerufen. Diese berechnet zuerst mithilfe der Koordinaten der Eckpunkte des verzerrten Rechtecks `eckPunkteBild` den Abstand `f` von Bild und optischem Zentrum (vgl. Kapitel 2.4) und anschließend die Koordinaten der Eckpunkte des Rechtecks im Original `eckPunkteOriginal`.

[4] In Kapitel 6.4 im Anhang befindet sich ein Klassendiagramm.

[5] Ausführliche Bedienungshinweise findet man in Kapitel 6.6 im Anhang.

Mithilfe dieser Punkte wird die Originalebene `originalEbene` definiert (vgl. Kapitel 2.5).

Als Nächstes wird die `createOriginalImage`-Methode aufgerufen. Das Ausgabebild entspricht dem Rechteck in der Originalebene. Für jedes Pixel wird die `bildPunkt`-Methode, die die Koordinaten des zugehörigen Bildpunkts zurückgibt, aufgerufen und die Farbe des dem Bildpunkt entsprechenden Pixels im Eingabebild übernommen. Falls ein geeignetes[6] Ausgabebild konstruiert werden konnte, wird dieses nun zurückgegeben.

3.1.3 Ebene

`Ebene`-Objekte (vgl. Quelltext 3) enthalten einen Stützvektor `stuetzVektor` und zwei Richtungsvektoren `richtungsVektor1` und `richtungsVektor2`. Somit enthalten sie alle Informationen, die man für eine Ebene in Parameterdarstellung benötigt.

Die `bildPunkt`-Methode gibt die x_1- und x_2-Koordinaten des Schnittpunkts der Ursprungsgeraden durch den Punkt mit dem Ortsvektor, der durch Einsetzen der Parameter aus der Ebenengleichung resultiert, mit der Bildebene zurück (vgl. Kapitel 2.6).

3.1.4 Vektor3

`Vektor3`-Objekte (vgl. Quelltext 4) entsprechen einem Vektor mit drei Komponenten. Die Komponenten werden im Attribut `xxx` gespeichert. Mittels der `get`-Methode können sie zurückgegeben werden. Außerdem gibt die `laenge`-Methode die Länge des Vektors zurück. Die statischen Methoden `produkt` und `summe` geben das Produkt von einem Vektor mit einem Skalar beziehungsweise die Summe zweier Vektoren zurück.

3.2 Hinweise zu den verwendeten Java-Klassen

Die Bilder, speziell auch `imgIn` und `imgOut` als Attribute von `Fenster` und `Korrektor`, sind Objekte der `BufferedImage`-Klasse[7]. Das hat den Vorteil, dass

[6] Das heißt, dass das Rechteck in der Originalebene echt sein muss. Manchmal erhält man aufgrund der Punktauswahl des Nutzers allerdings sehr große beziehungsweise sehr kleine Werte für f, λ_b, λ_c oder λ_d, sodass man von einem Fehler ausgehen muss. In diesem Fall wird `null` zurückgegeben und das `Fenster` lässt den Nutzer erneut die Eckpunkte des verzerrten Rechtecks auswählen (vgl. Kapitel 2.7).

[7] vgl. ORACLE: Class BufferedImage. Aus: `https://docs.oracle.com/javase/8/docs/api/java/awt/image/BufferedImage.html`, Stand:15.02.16

man sich über die `getRGB`-Methode die Farbe des Pixels mit den übergebenen Pixelkoordinaten zurückgeben lassen kann.

Um die Bilder für die grafische Darstellung zu skalieren und um gegebenenfalls das Ausgabebild zu skalieren, damit es dem DIN-A-Format entspricht, wird die `scale`-Methode eines Objektes der `Graphics2D`-Klasse[8] genutzt.

Um das Eingabebild aus einer Bilddatei einzulesen, beziehungsweise um das Ausgabebild zu exportieren, werden die `read`- und `write`-Methoden der `ImageIO`-Klasse[9] genutzt.

[8] vgl. ORACLE: Class Graphics2D. Aus: `https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html`, Stand: 20.02.16

[9] vgl. ORACLE: Reading/Loading an Image. Aus: `https://docs.oracle.com/javase/tutorial/2d/images/loadimage.html`, Stand: 23.02.16
und ORACLE: Writing/Saving an Image. Aus: `https://docs.oracle.com/javase/tutorial/2d/images/saveimage.html`, Stand: 23.02.16

4 Anwendung auf Beispiele

Die Java-Applikation wird mit verschiedenen Beispielen getestet. Dabei soll die Funktionsfähigkeit der Vorgehensweise überprüft und Schwierigkeiten des Programms aufgedeckt werden. Die Fotos wurden mit einer Nikon D5300 Spiegelreflexkamera mit einer Auflösung von 24,2 Megapixel aufgenommen.

4.1 Beispiele

Am Beispiel eines Gitters soll überprüft werden, inwieweit die Winkel und Proportionen des Originaldokuments nach der Korrektur der perspektivischen Verzerrung wiederhergestellt wurden. Mithilfe des Textdokuments soll die Lesbarkeit von Schrift nach der Korrektur der perspektivischen Verzerrung untersucht werden. Mit einem sehr schräg aufgenommenen Bild des Nummernschildes soll gezeigt werden, dass auch extrem perspektivisch verzerrte Schrift wieder lesbar gemacht werden kann.

4.1.1 Gitter

Auf dem Originaldokument Abbildung 9 ist ein Gitter mit quadratischen Kästchen abgebildet. Die Linien sind also alle entweder parallel oder stehen senkrecht zueinander und die Seiten der Kästchen sind alle gleich lang. Das Gitter wurde auf ein DIN-A4-Blatt ausgedruckt und schief fotografiert.

Beim Eingabebild Abbildung 13a mit einer leichten perspektivischen Verzerrung des Gitters sind im Ausgabebild Abbildung 13b die rechten Winkel alle wiederhergestellt worden und die Kästchen sind alle wieder gleich groß. Allerdings wurden sie leicht gestaucht, sodass sie nicht mehr quadratisch sind.

Ähnlich verhält sich das Ausgabebild Abbildung 14b des mäßig verzerrten Gitters aus dem Eingabebild Abbildung 14a. Hier sind die Kästchen sogar noch stärker gestaucht worden.

Das Ausgabebild Abbildung 15b des stark verzerrten Gitters aus Eingabebild Abbildung 15a stimmt zwar in den Winkeln und Längenverhältnissen gut mit dem Eingabebild überein, hier scheint allerdings eine andere Verzerrung hinzugekommen zu sein, da das Ausgabebild etwas zerknittert aussieht.

Ebenso ist auch das Ausgabebild Abbildung 16b des Gitters mit sehr starker perspektivischer Verzerrung aus dem Eingabebild Abbildung 16a verzerrt und sieht sogar stärker zerknittert aus. Weiterhin ist es zu einer Seite hin unscharf.

4.1.2 Textdokument

Das Originaldokument Abbildung 11b ist im DIN-A4-Format und enthält Text der Schriftart “Helvetia” in der Schriftgröße 12 pt. Nach der Korrektur der perspektivischen Verzerrung wurde hier die `scaleDinA`-Methode genutzt.

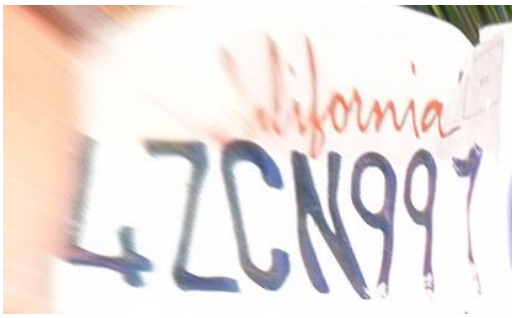
Das Dokument ist im Eingabebild Abbildung 10a leicht verzerrt. Das Ausgabebild Abbildung 10b ist dem Originaldokument sehr ähnlich und der Text ist an allen Stellen gut lesbar.

Ähnlich verhält sich das Ausgabebild Abbildung 11b des Textdokuments mit mäßiger perspektivischer Verzerrung aus Eingabebild Abbildung 11a. Dieses scheint allerdings wieder leicht zerknittert zu wirken.

Beim stark verzerrten Textdokument in Eingabebild Abbildung 12a wirkt das Ausgabebild Abbildung 12b zerknittert und die Schrift ist zwar entzifferbar, aber der obere Teil des Textdokuments, der im Eingabebild am weitesten hinten liegt, ist sehr unscharf.

4.1.3 Nummernschild

(a) Frontansicht



(b) Ausgabebild

(c) Eingabebild

Abbildung 2: Korrektur eines unlesbaren Nummernschildes

Die Frontansicht Abbildung 2a des Nummernschildes „4ZCN997” zeigt, dass das Nummernschild einem Rechteck mit abgerundeten Ecken entspricht. Aufgrund der

sehr starken perspektivischen Verzerrung des Nummernschildes im Eingabebild Abbildung 2c ist das Kennzeichen nicht mehr zu lesen. Die Schrift „4ZCN997" auf dem Ausgabebild Abbildung 2b ist hingegen gut erkennbar. Selbst der Bundesstaat „California" lässt sich noch entziffern.

Das Nummernschild ist auf dem Ausgabebild unscharf und immer noch verzerrt, nun allerdings nicht mehr perspektivisch. Es sieht aus, als wäre das Nummerschild gewölbt beziehungsweise verbogen.

4.2 Mögliche Fehlerursachen

Eine Ursache für die unterschiedlichen Seitenverhältnisse[10] des Rechtecks im Ausgabebild und Original könnte die ungenaue Auswahl der Eckpunkte des verzerrten Rechtecks durch den Benutzer sein, da diese mit der Mouse erfolgt. Weiterhin könnten Runden beziehungsweise das Abschneiden von Dezimalstellen durch das Programm und die Vernachlässigung anderer Eigenschaften der Kamera wegen des Modells der Lochkamera dazu beitragen. Letzteres könnte auch eine Rolle bei der neu auftretenden Verzerrung, die das Ausgabebild zerknittert oder verbogen wirken lässt, spielen. Außerdem sollte in Betracht gezogen werden, dass das Nummernschild leicht verbogen war und auch das Blatt mit dem Text beziehungsweise dem Gitter nicht vollständig glatt war, sodass diese im Gegensatz zu einem Rechteck nicht ganz eben sind. Die Berechnungen gehen jedoch von einem ebenen Rechteck aus.

Dass die Ausgabebilder nach der Korrektur der perspektivischen Verzerrung bei stark verzerrten Rechtecken teilweise unscharf werden, könnte zum einem daran liegen, dass sich die Fläche des Rechtecks im Eingabebild viel kleiner als die Fläche des Rechtecks im Ausgabebild ist. Somit ändert sich die Anzahl von Pixeln im korrigierten Bild, die eine Menge von Pixeln im Eingabebild mit dem verzerrten Rechteck repräsentieren. Zum anderen kommt hier auch ein technisches Problem der Kamera hinzu. Es ist schwer Dinge, die unterschiedlich weit von der Linse entfernt sind, gleichzeitig scharf zu stellen. Wenn das perspektivisch verzerrte Rechteck auf dem Eingabebild allerdings sehr schräg liegt, dann ist sein vorderer Rand viel näher an der Linse als der hintere.

[10] Um die Veränderung der Seitenverhältnisse nach der Korrektur der perspektivischen Verzerrung bei DIN-A-Formaten zu korrigieren, wurde die `scaleDinA`-Methode entwickelt, die das Ausgabebild entsprechend skaliert.

5 Fazit

Die Projektion vom Original zum Bild wurde umgekehrt, indem der Abstand vom Bild zum optischen Zentrum und die Koordinaten der Eckpunkte des Rechtecks im Original in Abhängigkeit von den Koordinaten der Bildpunkte berechnet wurden, womit dann eine Originalebene definiert wurde und schließlich jedem Punkt im Rechteck in der Originalebene ein Bildpunkt zugeordnet wurde. Mit einer Java-Applikation basierend auf diesen Schritten wurde die Vorgehensweise erfolgreich getestet.

Die Rechnungen zur Umkehr der Projektion vom Original zum Bild wurden weitestgehend mit dem Wissen aus dem Mathematikunterricht der Q1 durchgeführt, sodass Projektionsmatrizen, welche meist in der Literatur zum Thema vorkommen, nicht verwendet werden mussten.

Bilder mit einer leichten perspektivischen Verzerrung der betroffenen Rechtecke konnten so weit korrigiert werden, dass die Winkel und Längenverhältnisse des Originals fast vollständig wiederhergestellt wurden. Am Beispiel eines Nummernschildes wurde gezeigt, dass man mit dem entwickelten Programm auch Schriftzüge, die aufgrund von einer starken perspektivischen Verzerrung unlesbar geworden sind, wieder lesbar machen kann.

Allerdings lässt sich bei stark verzerrten Bildern wegen der Ungenauigkeit der Auswahl der Eckpunkte des verzerrten Rechtecks durch den Nutzer, dem Abschneiden von Dezimalstellen durch das Programm, dem Vernachlässigen weiterer Eigenschaften der Kamera und nicht vollständig ebenen Rechtecken im Original die perspektivische Verzerrung nicht komplett korrigieren. Weiterhin können beim korrigierten Bild andere Verzerrungen hinzukommen. Außerdem erscheinen verzerrte Teile des Bildes in der korrigierten Form etwas unscharf, da die wenigen Pixel, die das verzerrte Rechteck darstellen, beim korrigierten Rechteck von weitaus mehr Pixeln repräsentiert werden.

Des Weiteren steht eine Fallunterscheidung nach der Lage der Punkte des verzerrten Rechtecks, um die Einschränkungen aus Kapitel 2.7 aufzuheben, noch aus. So könnte man auch klären, ob es mögliche Koordinaten für die Eckpunkte des verzerrten Rechtecks gibt, bei denen es tatsächlich nicht möglich ist, das Rechteck im Original zu rekonstruieren.

Um in größeren Mengen Bilder mit dem Programm auszuwerten, wäre es hilfreich ein automatisches Erkennen der verzerrten Rechtecke zu ergänzen. Nach der Korrektur der perspektivischen Verzerrung könnte der Text mithilfe von automatischer Schrifterkennung weiter verarbeitet werden.

Mich persönlich hat es besonders überrascht, wie gut das Programm auch stark

perspektivisch verzerrte Rechtecke korrigieren konnte. Allerdings muss das perspektivisch verzerrte Rechteck auf dem Foto scharf sein. Dies hat mich beim Fotografieren des Nummernschildes einige Fehlversuche gekostet. Im Rahmen der Facharbeit ist mir klar geworden, dass ein Foto mehr Informationen enthalten kann, als man auf den ersten Blick sieht. Bei der Korrektur perspektivischer Verzerrung ist die Technik dem menschlichen Gehirn überlegen.

6 Anhang

6.1 Literatur

[1] BREDTHAUER, WILHEM/BRUNS, KLAUS GERD ET AL.: Impulse Physik+Chemie 5|6. Stuttgart 2008 (Klett), S.71

[2] JOHNSON, MICAH/FARID, HANY: Metric Measurements on a Plane from a Single Image. Aus: `http://www.mit.edu/~kimo/publications/metric/metric06.pdf`, Stand:10.02.16

[3] ORACLE: Class BufferedImage. Aus: `https://docs.oracle.com/javase/8/docs/api/java/awt/image/BufferedImage.html`, Stand:15.02.16

[4] ORACLE: Class Graphics2D. Aus: `https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html`, Stand: 20.02.16

[5] ORACLE: Reading/Loading an Image. Aus: `https://docs.oracle.com/javase/tutorial/2d/images/loadimage.html`, Stand: 23.02.16

[6] ORACLE: Writing/Saving an Image. Aus: `https://docs.oracle.com/javase/tutorial/2d/images/saveimage.html`, Stand: 23.02.16

6.2 Abbildungsverzeichnis

6.3 Lochkamera

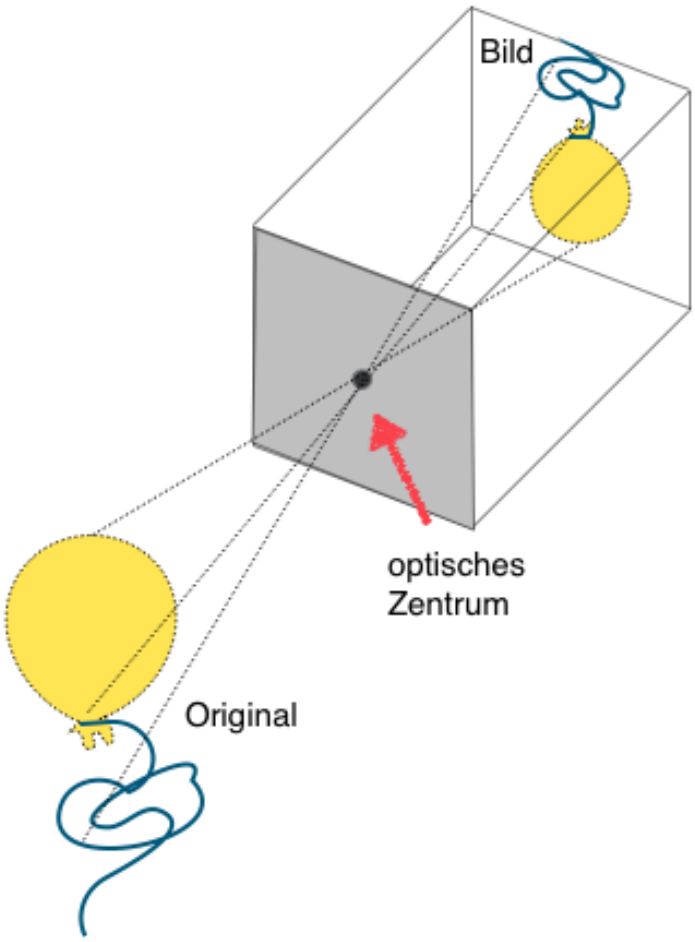


Abbildung 3: Lochkamera

Wenn Licht auf einen Gegenstand trifft, wird ein Anteil des Lichtes am Gegenstand gestreut. Von jedem Punkt auf dem Gegenstand verlaufen daher Lichtstrahlen in alle Richtungen. Trifft das Licht vom Gegenstand durch eine Blende auf einen Schirm, so entsteht dort für die Punkte auf dem Gegenstand je ein Lichtfleck. Die Lichtflecken setzen sich zu einem Bild des Gegenstandes zusammen, das seitenverkehrt ist und auf dem Kopf steht (vgl. Abbildung 6.3). Je kleiner das Loch in der Blende ist, umso kleiner werden die Lichtflecken, sodass sie sich weniger überlappen und das Bild schärfer wird.[11]

Beim verwendeten Modell der Lochkamera in Kapitel 2 wird das Loch als Punkt angenommen, sodass von jedem Punkt des Rechtecks im Original beziehungsweise jedem Pixel im Ausgabebild genau ein Lichtstrahl durch das Loch fällt, der einen Bildpunkt im verzerrten Rechteck auf dem Bild erzeugt.

[11] vgl. BREDTHAUER, WILHEM/BRUNS, KLAUS GERD U. A.: Impulse Physik+Chemie 5|6. Stuttgart 2008 (Klett), S.71

6.4 Klassendiagramm

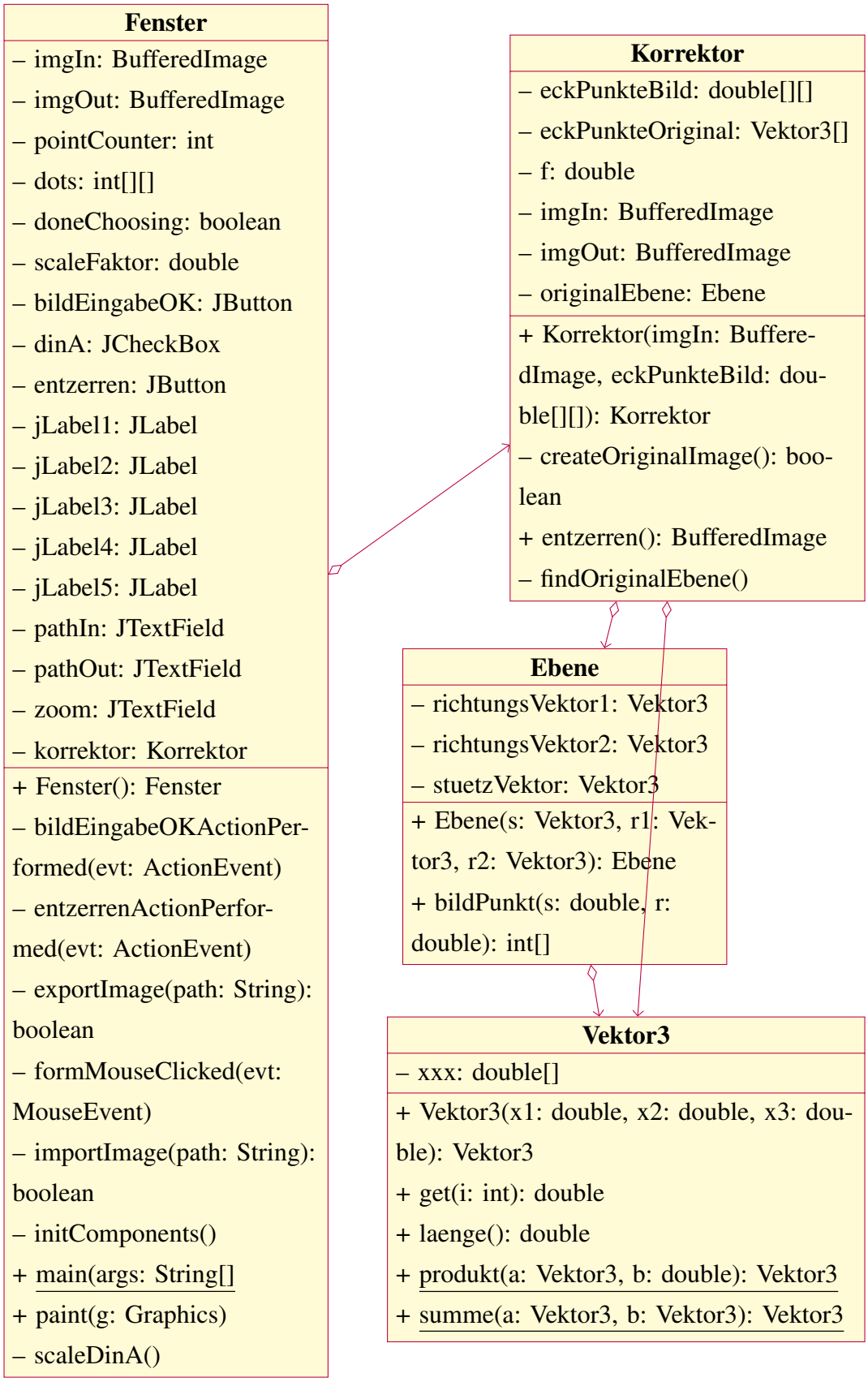


Abbildung 4: Klassendiagramm der Java-Applikation

6.5 Quelltext der Java-Applikation

Quelltextverzeichnis

Quelltext 1: Fenster Klasse: Grafische Darstellung und Ein- und Ausgabeverarbeitung

```
1  package rectification;
2
3  import java.awt.BasicStroke;
4  import java.awt.Color;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7  import java.awt.image.BufferedImage;
8  import java.io.File;
9  import java.io.IOException;
10 import java.util.logging.Level;
11 import java.util.logging.Logger;
12 import javax.imageio.ImageIO;
13
14 /**
15  *
```

```
16  * @author hannah
17  */
18 public class Fenster extends javax.swing.JFrame {
19
20     /**
21      * Creates new form Fenster
22      */
23     public Fenster() {
24         initComponents();
25         //Komponenten, die erst nach Bildeingabe sichtbar sein sollen
26         jLabel2.setVisible(false);
27         entzerren.setVisible(false);
28         jLabel4.setVisible(false);
29         jLabel5.setVisible(false);
30         pathOut.setVisible(false);
31         dinA.setVisible(false);
32         scaleFaktor=Double.parseDouble(zoom.getText());
33         /*damit noch nicht gesetzte Punkte außerhalb des sichtbaren gezeichnet werden*/
34         dots = new int[][]{{-5,-5},{-5,-5},{-5,-5},{-5,-5}};
35         pointCounter=5; //Punktauswahl noch nicht möglich
36     }
37
38     /**
39      * This method is called from within the constructor to initialize the form.
40      * WARNING: Do NOT modify this code. The content of this method is always
```

```
41      * regenerated by the Form Editor.
42      */
43     @SuppressWarnings("unchecked")
44     // <editor-fold defaultstate="collapsed" desc="Generated Code">
45     private void initComponents() {
46
47         pathIn = new javax.swing.JTextField();
48         bildEingabeOK = new javax.swing.JButton();
49         entzerren = new javax.swing.JButton();
50         jLabel1 = new javax.swing.JLabel();
51         jLabel2 = new javax.swing.JLabel();
52         zoom = new javax.swing.JTextField();
53         jLabel3 = new javax.swing.JLabel();
54         pathOut = new javax.swing.JTextField();
55         jLabel4 = new javax.swing.JLabel();
56         dinA = new javax.swing.JCheckBox();
57         jLabel5 = new javax.swing.JLabel();
58
59         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
60         setResizable(false);
61         addMouseListener(new java.awt.event.MouseAdapter() {
62             public void mouseClicked(java.awt.event.MouseEvent evt) {
63                 formMouseClicked(evt);
64             }
65         });
```

```
66
67          bildEingabeOK.setText("OK");
68          bildEingabeOK.addActionListener(new java.awt.event.ActionListener() {
69              public void actionPerformed(java.awt.event.ActionEvent evt) {
70                  bildEingabeOKActionPerformed(evt);
71              }
72          });
73
74          entzerren.setText("Entzerren");
75          entzerren.setEnabled(false);
76          entzerren.addActionListener(new java.awt.event.ActionListener() {
77              public void actionPerformed(java.awt.event.ActionEvent evt) {
78                  entzerrenActionPerformed(evt);
79              }
80          });
81
82          jLabel1.setText("Filename:");
83
84          jLabel2.setText("Eckpunkte eines verzerrten Rechtecks entgegen des");
85
86          zoom.setText("1");
87
88          jLabel3.setText("Zoom:");
89
90          pathOut.setText("bild.png");
```

```
91
92          jLabel4.setText("Speichern unter:");
93
94          dinA.setText("DIN ");
95
96          jLabel5.setText("Uhrzeigersinns auswählen (links unten beginnen)");
97
98          javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
99          getContentPane().setLayout(layout);
100         layout.setHorizontalGroup(
101             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
102             .addGroup(layout.createSequentialGroup()
103                 .addContainerGap()
104                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
105                     .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
106                         .addGroup(layout.createSequentialGroup()
107                             .addComponent(jLabel4)
108                             .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
109                             .addComponent(pathOut, javax.swing.GroupLayout.PREFERRED_SIZE, 83, javax.swing.
                                GroupLayout.PREFERRED_SIZE)
110                             .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
111                             .addComponent(dinA, javax.swing.GroupLayout.PREFERRED_SIZE, 60, javax.swing.
                                GroupLayout.PREFERRED_SIZE)
112                             .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.
                                GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
```

```
113                         .addComponent(entzerren, javax.swing.GroupLayout.PREFERRED_SIZE, 84, javax.swing.
                               GroupLayout.PREFERRED_SIZE))
114                     .addComponent(jLabel2))
115                 .addGroup(layout.createSequentialGroup()
116                     .addComponent(jLabel1)
117                     .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
118                     .addComponent(pathIn, javax.swing.GroupLayout.PREFERRED_SIZE, 92, javax.swing.
                           GroupLayout.PREFERRED_SIZE)
119                     .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
120                     .addComponent(jLabel3)
121                     .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
122                     .addComponent(zoom, javax.swing.GroupLayout.PREFERRED_SIZE, 46, javax.swing.
                           GroupLayout.PREFERRED_SIZE)
123                     .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
124                     .addComponent(bildEingabeOK, javax.swing.GroupLayout.PREFERRED_SIZE, 50, javax.swing.
                           GroupLayout.PREFERRED_SIZE))
125                 .addComponent(jLabel5))
126             .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
127         );
128         layout.setVerticalGroup(
129             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
130             .addGroup(layout.createSequentialGroup()
131                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
132                     .addComponent(pathIn, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.
                           DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
133                     .addComponent(bildEingabeOK)
134                     .addComponent(jLabel1)
135                     .addComponent(zoom, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.
                            DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
136                     .addComponent(jLabel3))
137                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
138                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
139                     .addComponent(jLabel4)
140                     .addComponent(pathOut, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.
                            DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
141                     .addComponent(dinA)
142                     .addComponent(entzerren))
143                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
144                 .addComponent(jLabel2)
145                 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
146                 .addComponent(jLabel5)
147                 .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
148         );
149 
150         pack();
151     }// </editor-fold>
152 
153     private void bildEingabeOKActionPerformed(java.awt.event.ActionEvent evt) {
154         /*Bild einlesen und Komponenten für die nächsten Schritte aktivieren*/
155         scaleFaktor= Double.parseDouble(zoom.getText());
```

```
156         pathIn.setEditable(!importImage(pathIn.getText()));
157         zoom.setEditable(pathIn.isEditable());
158         bildEingabeOK.setEnabled(pathIn.isEditable());
159         jLabel2.setVisible(!pathIn.isEditable());
160         entzerren.setVisible(!pathIn.isEditable());
161         jLabel4.setVisible(!pathIn.isEditable());
162         jLabel5.setVisible(!pathIn.isEditable());
163         pathOut.setText(pathIn.getText());
164         pathOut.setVisible(!pathIn.isEditable());
165         dinA.setVisible(!pathIn.isEditable());
166         pointCounter=0;
167         repaint();
168     }
169 
170     private void formMouseClicked(java.awt.event.MouseEvent evt) {
171         /*Punktauswahl auf Eingabebild, Koordinaten als Bildschirm-Koordinaten des Eingabebildes relativ zur
               linken oberen Ecke des Eingabebildes*/
172         if (evt.getX() > 0 && evt.getY() > 140 && evt.getX() < imgIn.getWidth()*scaleFaktor && evt.getY() -
               140 < imgIn.getHeight()*scaleFaktor) {
173             if(pointCounter<4){
174                 dots[pointCounter][0] = evt.getX();
175                 dots[pointCounter][1] = evt.getY()-140;
176                 pointCounter++;
177                 if(pointCounter==4){
178                     doneChoosing=true;
```

```
179                     entzerren.setEnabled(true);
180                 }
181             }
182         }
183         repaint();
184     }
185 
186     private void entzerrenActionPerformed(java.awt.event.ActionEvent evt) {
187         /*Eckpunkte des Rechtecks im Bild nach Punktauswahl, Koordinaten relativ zum Bildmittelpunkt*/
188         double[][] eckPunkteBild = new double [4][2];
189         for (int i = 0; i < 4; i++) {
190             eckPunkteBild[i][0] = dots[i][0]/scaleFaktor - imgIn.getWidth()/2.0;
191             eckPunkteBild[i][1] = -imgIn.getHeight()/2.0+dots[i][1]/scaleFaktor;
192         }
193         //Entzerren des Eingabebildes mithilfe des Korrektors
194         korrektor= new Korrektor(imgIn, eckPunkteBild);
195         imgOut=korrektor.entzerren();
196         /*Ausgabe des Ausgabebildes und Deaktivierung der Eingabegelegenheiten, wenn sinnvolles Ausgabebild
               entstanden, sonst Zurücksetzten aller Eingaben nach Bildeingabe*/
197         if(imgOut!=null){
198             scaleDinA();
199             exportImage(pathOut.getText());
200             entzerren.setEnabled(false);
201             pathOut.setEditable(false);
202             dinA.setEnabled(false);
```

```
203             repaint();
204             this.setSize((int)(imgIn.getWidth()*scaleFaktor+ imgOut.getWidth()*scaleFaktor+20), (int)(Math.
                    max(imgIn.getHeight()*scaleFaktor, imgOut.getHeight()*scaleFaktor)+140));
205             System.out.println("done");
206         }
207         else {
208             System.out.println("incompatible point selection");
209             doneChoosing=false;
210             pointCounter=0;
211             dots = new int[][]{{-5,-5},{-5,-5},{-5,-5},{-5,-5}};
212             repaint();
213         }
214     }
215 
216     /**
217      * @param args the command line arguments
218      */
219     public static void main(String args[]) {
220         /* Set the Nimbus look and feel */
221         //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
222         /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
223          * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
224          */
225         try {
226             for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()
```

```
            ) {
227             if ("Nimbus".equals(info.getName())) {
228                 javax.swing.UIManager.setLookAndFeel(info.getClassName());
229                 break;
230             }
231         }
232     } catch (ClassNotFoundException ex) {
233         java.util.logging.Logger.getLogger(Fenster.class.getName()).log(java.util.logging.Level.SEVERE,
                null, ex);
234     } catch (InstantiationException ex) {
235         java.util.logging.Logger.getLogger(Fenster.class.getName()).log(java.util.logging.Level.SEVERE,
                null, ex);
236     } catch (IllegalAccessException ex) {
237         java.util.logging.Logger.getLogger(Fenster.class.getName()).log(java.util.logging.Level.SEVERE,
                null, ex);
238     } catch (javax.swing.UnsupportedLookAndFeelException ex) {
239         java.util.logging.Logger.getLogger(Fenster.class.getName()).log(java.util.logging.Level.SEVERE,
                null, ex);
240     }
241     //</editor-fold>
242 
243     /* Create and display the form */
244     java.awt.EventQueue.invokeLater(new Runnable() {
245         public void run() {
246             new Fenster().setVisible(true);
```

```
247             }
248         });
249     }
250 
251     //Parameter Deklaration für Bildbearbeitung
252     private BufferedImage imgIn, imgOut;
253     private int pointCounter;
254     private int[][] dots;
255     private boolean doneChoosing;
256     private double scaleFaktor;
257     private Korrektor korrektor;
258 
259     private boolean importImage(String path) {
260         //einlesen eines des Eingabebildes
261         try {
262             imgIn = ImageIO.read(new File("BeispieleEin/"+path));
263         } catch (IOException e) {
264             System.out.println("incorrect path given");
265             return false;
266         }
267         setSize((int)(imgIn.getWidth()*scaleFaktor), (int)(imgIn.getHeight()*scaleFaktor + 140));
268         return true;
269     }
270     private boolean exportImage(String path){
271         //ausgeben des Ausgabebildes
```

```
272         try {
273                 ImageIO.write(imgOut, "png", new File("BeispieleAus/"+path));
274             } catch (IOException ex) {
275                 Logger.getLogger(Fenster.class.getName()).log(Level.SEVERE, null, ex);
276                 return false;
277             }
278         return true;
279     }
280     private void scaleDinA(){
281         //Skalieren des Ausgabebildes für DinA Formate
282         if(dinA.isSelected()){
283                 BufferedImage imgTemp=imgOut;
284                 int w=imgTemp.getWidth(), h=imgTemp.getHeight();
285                 if(imgTemp.getWidth()>imgTemp.getHeight()) w=(int)(imgTemp.getHeight()*Math.sqrt(2));
286                 else h=(int)(imgTemp.getWidth()*Math.sqrt(2));
287                 imgOut= new BufferedImage(w,h,BufferedImage.TYPE_INT_ARGB);
288                 Graphics2D g2=imgOut.createGraphics();
289                 g2.scale((double)(w)/imgTemp.getWidth(), (double)(h)/imgTemp.getHeight());
290                 g2.drawImage(imgTemp,0,0,null);
291                 g2.dispose();
292         }
293     }
294
295     @Override
296     public void paint(Graphics g) {
```

```
297         super.paint(g);
298         Graphics2D g2=(Graphics2D) g;
299         //Darstellung des Eingabebildes
300         if (imgIn != null) {
301             g2.scale(scaleFaktor, scaleFaktor);
302             g2.drawImage(imgIn, 0, (int) (140/scaleFaktor), null);
303             g2.scale(1/scaleFaktor, 1/scaleFaktor);
304         }
305         //Darstellung der ausgewählten Punkte
306         if (!doneChoosing) {
307             for (int[] d : dots) {
308                 g2.setColor(Color.red);
309                 g2.fillOval(d[0], d[1]+140, 5, 5);
310             }
311         }
312         //Darstellung des ausgewählten Vierecks
313         if(doneChoosing) {
314             g2.setColor(Color.red);
315             g2.setStroke(new BasicStroke(5));
316             g2.drawPolygon(new int[]{dots[0][0], dots[1][0], dots[2][0], dots[3][0]}, new int[]{dots
                    [0][1]+140, dots[1][1]+140, dots[2][1]+140, dots[3][1]+140}, 4);
317         }
318         //Darstellung des Ausgabebildes
319         if(imgOut!= null) {
320             g2.scale(scaleFaktor, scaleFaktor);
```

```
321             g2.drawImage(imgOut, (int) (imgIn.getWidth()+20/scaleFaktor), (int) (140/scaleFaktor) ,  null);
322             g2.scale(1/scaleFaktor, 1/scaleFaktor);
323         }
324         g2.dispose();
325     }
326     // Variables declaration - do not modify
327     private javax.swing.JButton bildEingabeOK;
328     private javax.swing.JCheckBox dinA;
329     private javax.swing.JButton entzerren;
330     private javax.swing.JLabel jLabel1;
331     private javax.swing.JLabel jLabel2;
332     private javax.swing.JLabel jLabel3;
333     private javax.swing.JLabel jLabel4;
334     private javax.swing.JLabel jLabel5;
335     private javax.swing.JTextField pathIn;
336     private javax.swing.JTextField pathOut;
337     private javax.swing.JTextField zoom;
338     // End of variables declaration
339 }
```

Quelltext 2: Korrektor Klasse: Berechnung der Orignal-Koordinaten und Herstellung des Ausgabebildes

```
1 package rectification;
2
3 import java.awt.image.BufferedImage;
4
```

```
5  /**
6   *
7   * @author hannah
8   */
9  //Entzerrung des Eingabebildes
10 public class Korrektor {
11     //Paramter Deklaration
12     private Ebene originalEbene;
13     private Vektor3[] eckPunkteOriginal;
14     private double[][] eckPunkteBild;
15     private BufferedImage imgOut;
16     private BufferedImage imgIn;
17     private double f; //Abstand von Bild und optischem Zentrum
18     //Konstruktor
19     public Korrektor(BufferedImage imgIn, double[][] eckPunkteBild){
20         this.imgIn=imgIn;
21         this.eckPunkteBild=eckPunkteBild;
22     }
23     //Eingabebild entzerren und zurückgeben
24     public BufferedImage entzerren(){
25         findOriginalEbene();
26         if(createOriginalImage()) return imgOut;
27         return null;
28     }
29     /*Ermitteln der Originalebene in dem Eckpunkte des Originalbildes berrechnet werden*/
```

```
30      private void findOriginalEbene() {
31          double  a1 = eckPunkteBild[0][0], a2 = eckPunkteBild[0][1], b1 = eckPunkteBild[1][0], b2 =
                eckPunkteBild[1][1], lb, c1 = eckPunkteBild[2][0], c2 = eckPunkteBild[2][1], lc, d1 = eckPunkteBild
                [3][0], d2 = eckPunkteBild[3][1], ld;
32          lb=((a2-d2)*(c1-d1)-(d1-a1)*(d2-c2))/((b1-d1)*(d2-c2)-(d2-b2)*(c1-d1));
33          lc=(d1+lb*(b1-d1)-a1)/(c1-d1);
34          ld=1-lb+lc;
35          f=Math.sqrt(((lb*b1-a1)*(ld*d1-a1)+(lb*b2-a2)*(ld*d2-a2))/((lb-1)*(1-ld)));
36          Vektor3 a=new Vektor3(a1,a2,f), c=new Vektor3(c1,c2,f), b=new Vektor3(b1,b2,f), d=new Vektor3(d1,d2,f)
                ;
37          Vektor3 x=Vektor3.summe(Vektor3.produkt(b, lb), Vektor3.produkt(a, -1)), y=Vektor3.summe(Vektor3.
                produkt(d, ld), Vektor3.produkt(a, -1));
38          x=Vektor3.produkt(x, 1.0/x.laenge());
39          y=Vektor3.produkt(y, 1.0/y.laenge());
40          originalEbene= new Ebene(a,x,y);
41          eckPunkteOriginal=new Vektor3[]{a,Vektor3.produkt(b, lb), Vektor3.produkt(c, lc), Vektor3.produkt(d,
                ld)};
42      }
43      /*Herstellung des Ausgabebildes indem jedem Pixel im Ausgabebild ein Pixel im Eingabebild zugeordnet wird.
            Rückgabe eines Wahrheitswerts, um zu kennzeichnen, ob ein geeignetes Originalbild möglich ist (Bildflä
            che>0)*/
44      private boolean createOriginalImage(){
45          int w=(int) Vektor3.summe(Vektor3.produkt(eckPunkteOriginal[0], -1), eckPunkteOriginal[1]).laenge();
46          int h=(int) Vektor3.summe(Vektor3.produkt(eckPunkteOriginal[0], -1), eckPunkteOriginal[3]).laenge();
47          if(w*h==0) return false;
```

```
48          imgOut=new BufferedImage(w,h, BufferedImage.TYPE_INT_ARGB);
49          for(int x=0; x<w; x++){
50              for(int y=0; y<h; y++){
51                  int[] p=originalEbene.bildPunkt(x, imgOut.getHeight()-y);
52                  imgOut.setRGB(x, y, imgIn.getRGB(p[0]+imgIn.getWidth()/2, p[1]+imgIn.getHeight()/2));
53              }
54          }
55          return true;
56      }
57  }
```

Quelltext 3: Ebene Klasse: Ebene in Parameterdarstellung mit Schnittpunkt-Berechnung

```
1   package rectification;
2   /**
3    *
4    * @author hannah
5    */
6   //Ebene in Parameterdarstellung
7   public class Ebene {
8       //Parameter Deklaration
9       private final Vektor3 stuetzVektor;
10      private final Vektor3 richtungsVektor1;
11      private final Vektor3 richtungsVektor2;
12      //Konstruktor
13      public Ebene(Vektor3 s, Vektor3 r1, Vektor3 r2){
```

```
14          if(s==null) s= new Vektor3(0,0,0);
15          stuetzVektor=s;
16          richtungsVektor1=Vektor3.produkt(r1, 1/r1.laenge());
17          richtungsVektor2=Vektor3.produkt(r2, 1/r2.laenge());
18      }
19      /*Aufruf mit Parametern für Ebenengleichung um Punkt zu beschreiben, Rückgabe der x1- und x2-Koordinate
            des Schnittpunkts der Ursprungsgeraden durch Punkt mit Bild*/
20      public int[] bildPunkt(double s, double r){
21          double a1=stuetzVektor.get(0),a2=stuetzVektor.get(1), f=stuetzVektor.get(2), x1=richtungsVektor1.get(0
                ), x2=richtungsVektor1.get(1),x3=richtungsVektor1.get(2), y1=richtungsVektor2.get(0), y2=
                richtungsVektor2.get(1), y3=richtungsVektor2.get(2);
22          int p1, p2;
23          p1 = (int) (f*(a1+s*x1+r*y1)/(f+s*x3+r*y3));
24          p2 = (int) (f*(a2+s*x2+r*y2)/(f+s*x3+r*y3));
25          return new int[]{p1,p2};
26      }
27  }
```

Quelltext 4: Vektor3 Klasse: Vektor mit drei Komponenten

```
1   package rectification;
2   /**
3    *
4    * @author hannah
5    */
6   //Vektor mit drei Komponenten
```

```java
public class Vektor3 {
    //Parameter Deklaration
    private double[] xxx;
    //Konstruktor
    public Vektor3(double x1, double x2, double x3){
        xxx = new double[]{x1, x2, x3};
    }
    //Rückgabe einer Komponente
    public double get(int i){
        return xxx[i];
    }
    //Rückgabe der Länge des Vektors
    public double laenge(){
        return Math.sqrt(Math.pow(xxx[0], 2)+Math.pow(xxx[1], 2)+Math.pow(xxx[2], 2));
    }
    //Rückgabe der Summe zweier Vektoren
    public static Vektor3 summe(Vektor3 a, Vektor3 b){
        return new Vektor3(a.get(0)+b.get(0),a.get(1)+b.get(1),a.get(2)+b.get(2));
    }
    //Rückgabe des Produkts von Vektor mit Skalar
    public static Vektor3 produkt(Vektor3 a, double b){
        return new Vektor3(a.get(0)*b, a.get(1)*b, a.get(2)*b);
    }
}
```

6.6 Bedienungshinweise für das Programm

1. Der Dateiname[12] des Eingabebildes wird eingegeben (vgl. Abbildung 5). Zusätzlich kann ein Skalierungsfaktor eingegeben werden, um die Größe der Darstellung zu bestimmen. Ist der Dateiname ungültig, so wird dies in der Konsole angezeigt und man darf ihn ändern und es erneut versuchen.

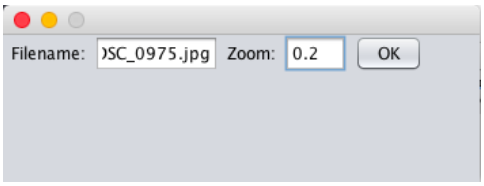


Abbildung 5: Eingabe des Eingabebildes

2. Die Eckpunkte eines verzerrten Reckecks werden durch Anklicken ausgewählt. Dies sollte links unten beginnend gegen den Uhrzeigersinn erfolgen (vgl. Abbildung 6). Anschließend kann man einen Dateinamen zur Speicherung[13] des Ausgabebildes angeben. Das Ausgabebild kann gegebenenfalls skaliert werde, um dem DIN-A-Format zu entsprechen.

Abbildung 6: Auswahl der Eckpunkte des verzerrten Recktecks

[12]Das Eingabebild muss sich im Projektordner im Ordner „BeispieleEin" befinden und möglichst unbearbeitet vorliegen.

[13]Das Ausgabebild wird in den Ordner „BeispieleAus" im Projektordner exportiert.

3. Das Ausgabebild wird nun dargestellt (vgl. Abbildung 7). Für Details kann man es nun im Ordner „BeispieleAus” öffnen. Lässt sich mit den gewählten Eckpunkten kein geeignetes Ausgabebild konstruieren, so wird dies in der Konsole ausgegeben und man muss erneut Eckpunkte auswählen.

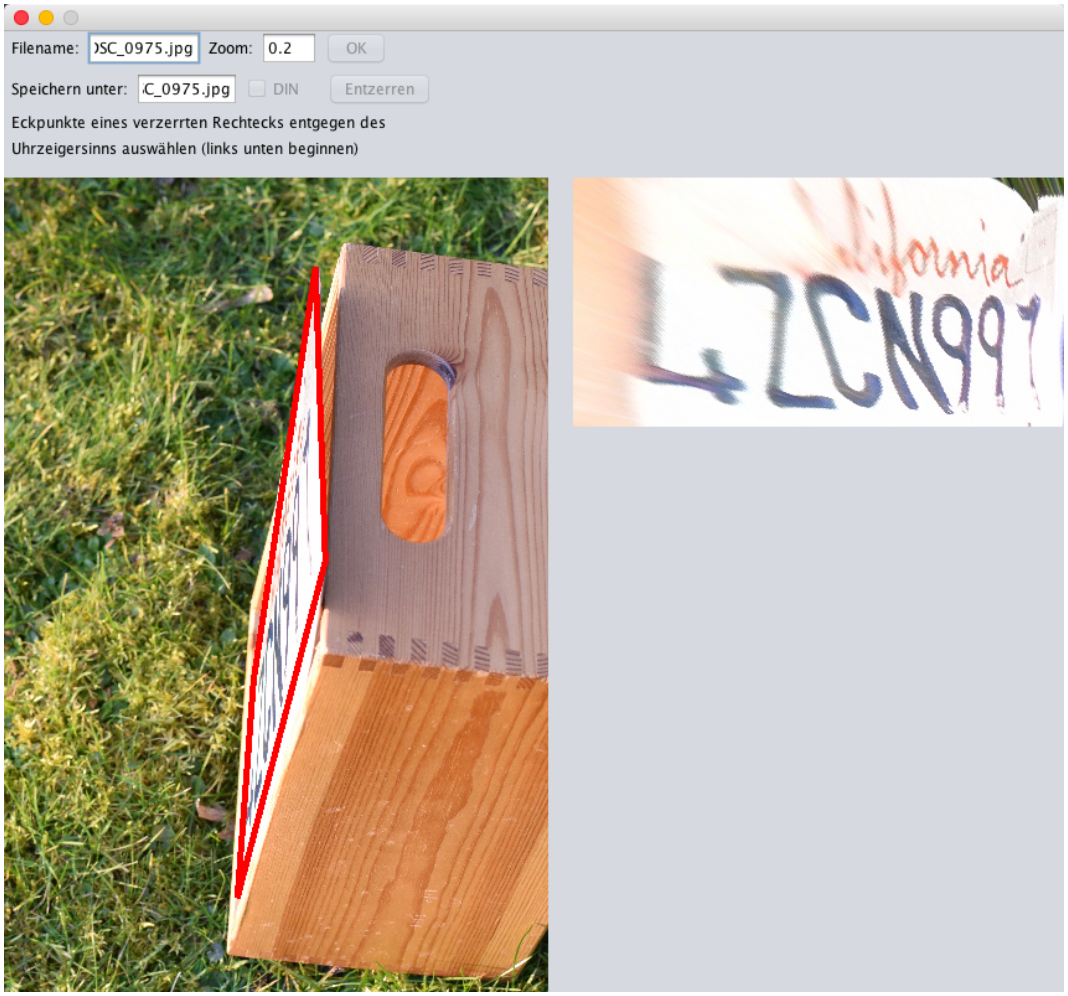
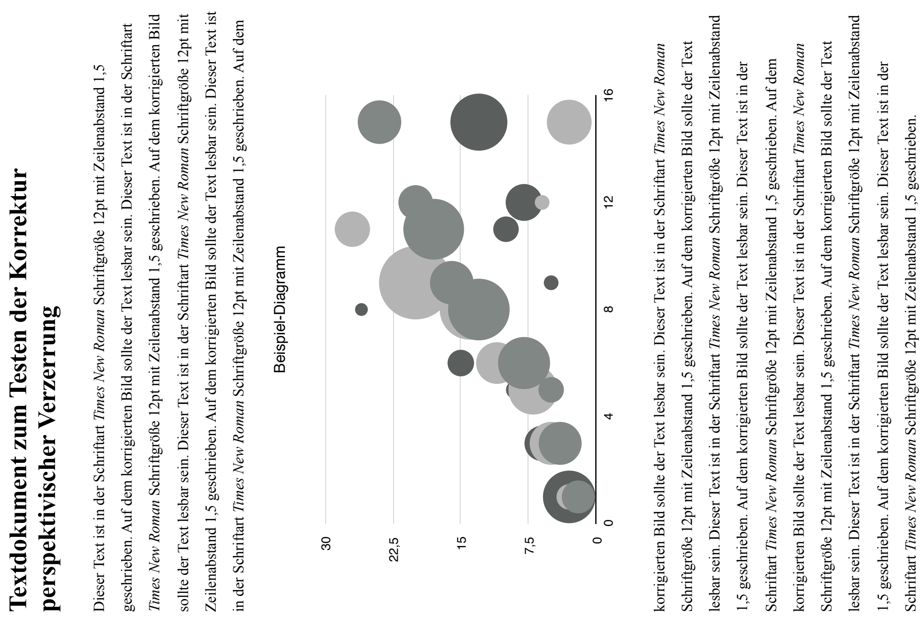


Abbildung 7: Darstellung des Ausgabebildes

6.7 Bildbeispiele



09. März 2016

Textdokument zum Testen der Korrektur perspektivischer Verzerrung

Dieser Text ist in der Schriftart *Times New Roman* Schriftgröße 12pt mit Zeilenabstand 1,5 geschrieben. Auf dem korrigierten Bild sollte der Text lesbar sein. Dieser Text ist in der Schriftart *Times New Roman* Schriftgröße 12pt mit Zeilenabstand 1,5 geschrieben. Auf dem korrigierten Bild sollte der Text lesbar sein. Dieser Text ist in der Schriftart *Times New Roman* Schriftgröße 12pt mit Zeilenabstand 1,5 geschrieben. Auf dem korrigierten Bild sollte der Text lesbar sein. Dieser Text ist in der Schriftart *Times New Roman* Schriftgröße 12pt mit Zeilenabstand 1,5 geschrieben. Auf dem

korrigierten Bild sollte der Text lesbar sein. Dieser Text ist in der Schriftart *Times New Roman* Schriftgröße 12pt mit Zeilenabstand 1,5 geschrieben. Auf dem korrigierten Bild sollte der Text lesbar sein. Dieser Text ist in der Schriftart *Times New Roman* Schriftgröße 12pt mit Zeilenabstand 1,5 geschrieben. Auf dem korrigierten Bild sollte der Text lesbar sein. Dieser Text ist in der Schriftart *Times New Roman* Schriftgröße 12pt mit Zeilenabstand 1,5 geschrieben. Auf dem korrigierten Bild sollte der Text lesbar sein. Dieser Text ist in der Schriftart *Times New Roman* Schriftgröße 12pt mit Zeilenabstand 1,5 geschrieben. Auf dem korrigierten Bild sollte der Text lesbar sein. Dieser Text ist in der Schriftart *Times New Roman* Schriftgröße 12pt mit Zeilenabstand 1,5 geschrieben. Auf dem korrigierten Bild sollte der Text lesbar sein. Dieser Text ist in der Schriftart *Times New Roman* Schriftgröße 12pt mit Zeilenabstand 1,5 geschrieben.

Abbildung 8: Textdokument: Original

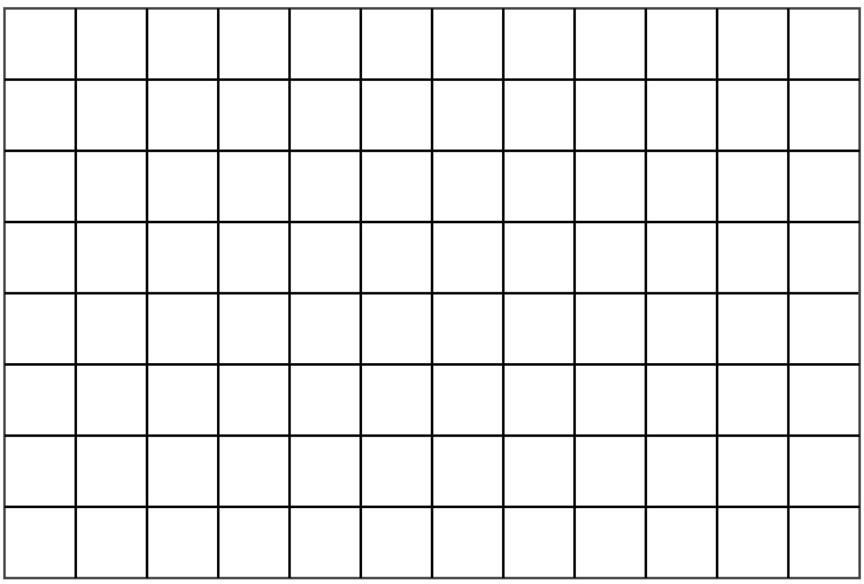
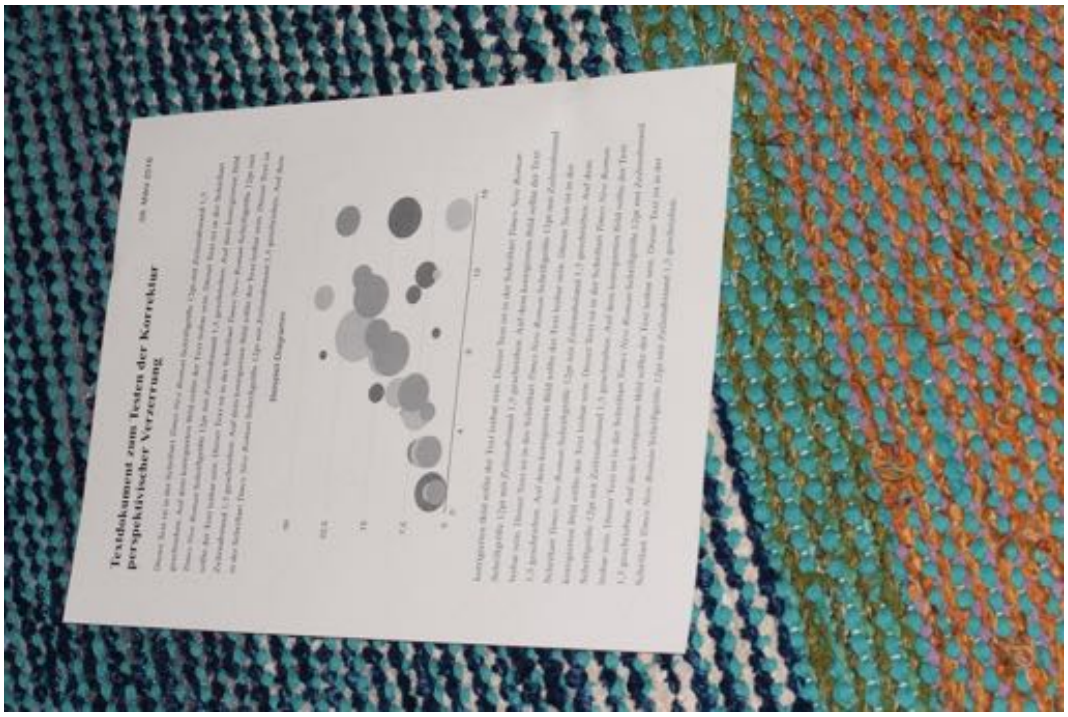
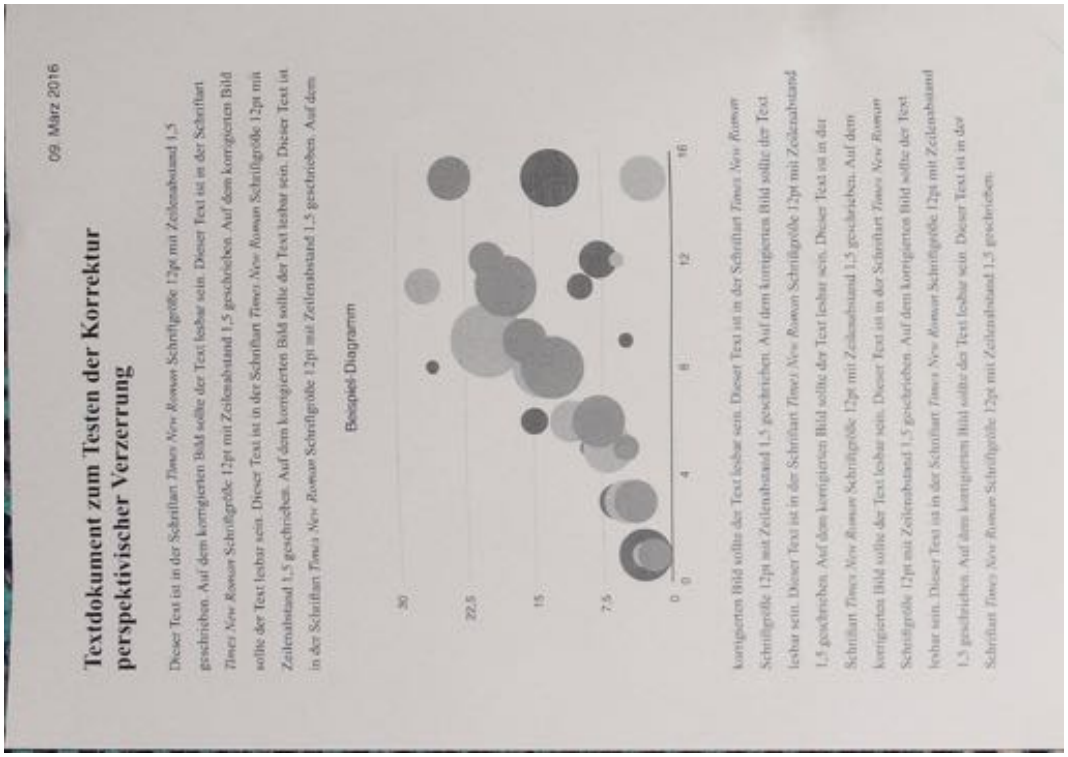


Abbildung 9: Gitter: Original



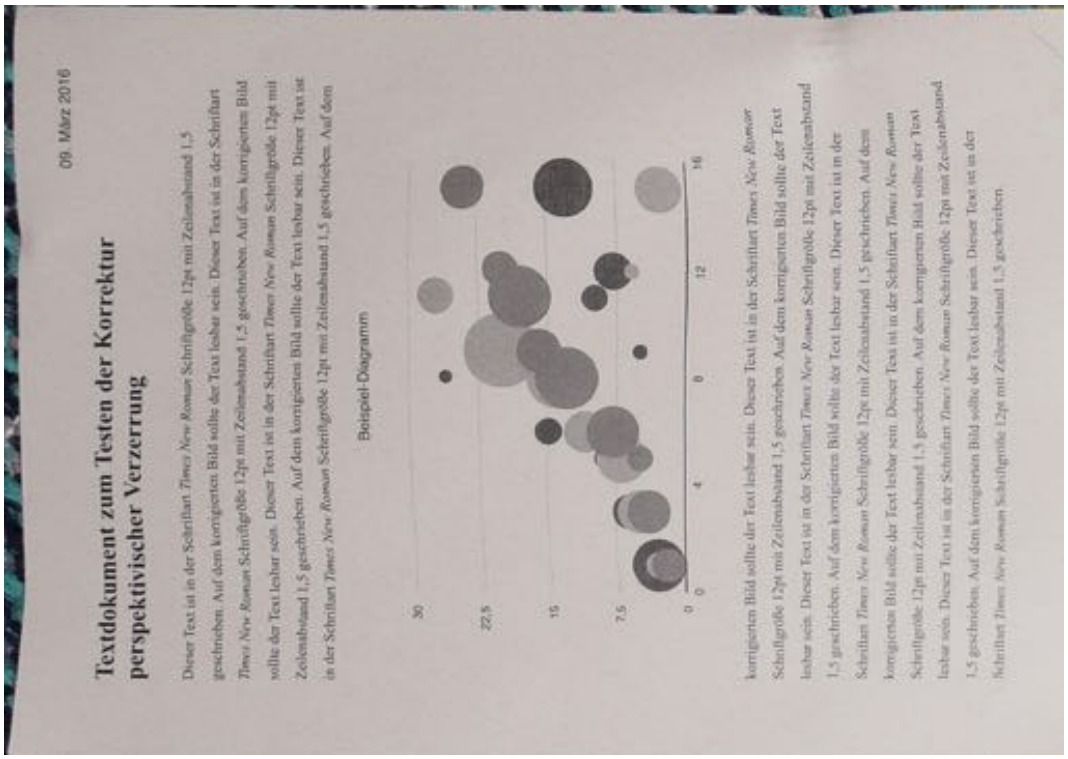
(a) Eingabebild



(b) Ausgabebild

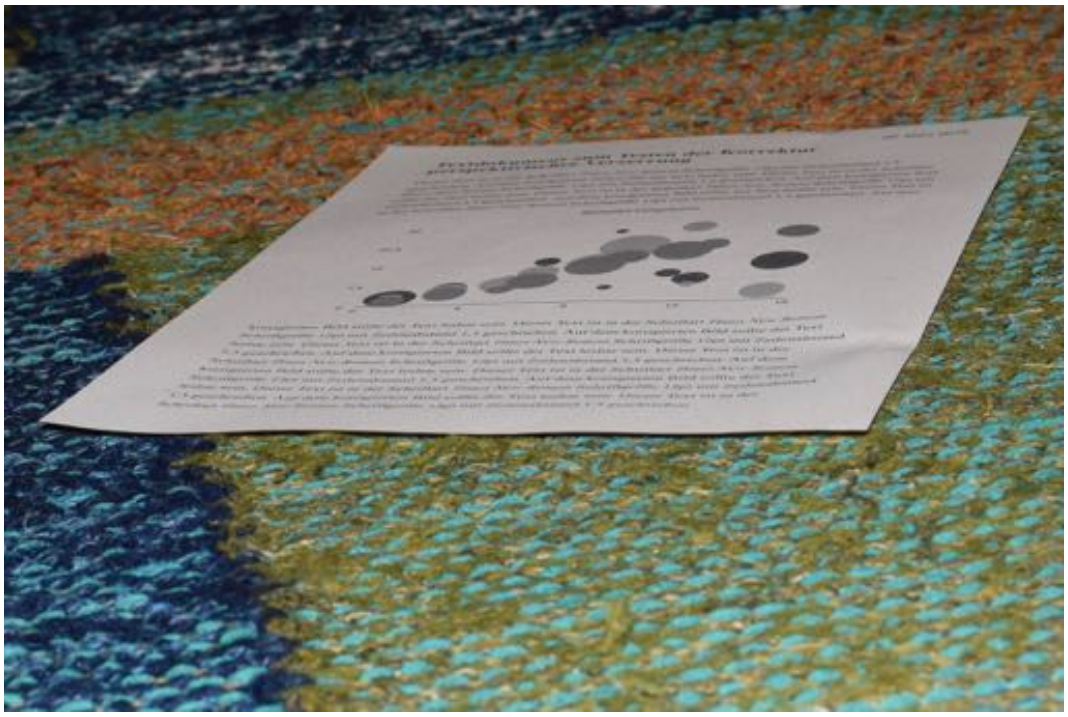
Abbildung 10: Textdokument (leicht verzerrt)

(a) Eingabebild

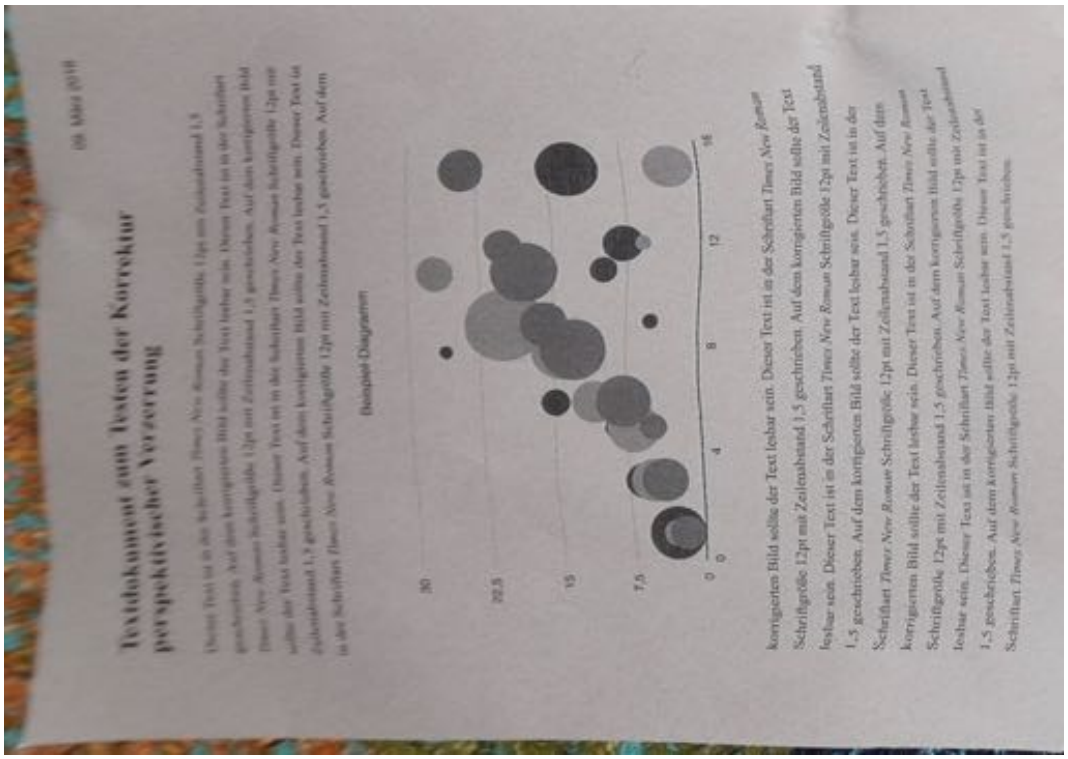


(b) Ausgabebild

Abbildung 11: Textdokument (mäßig verzerrt)

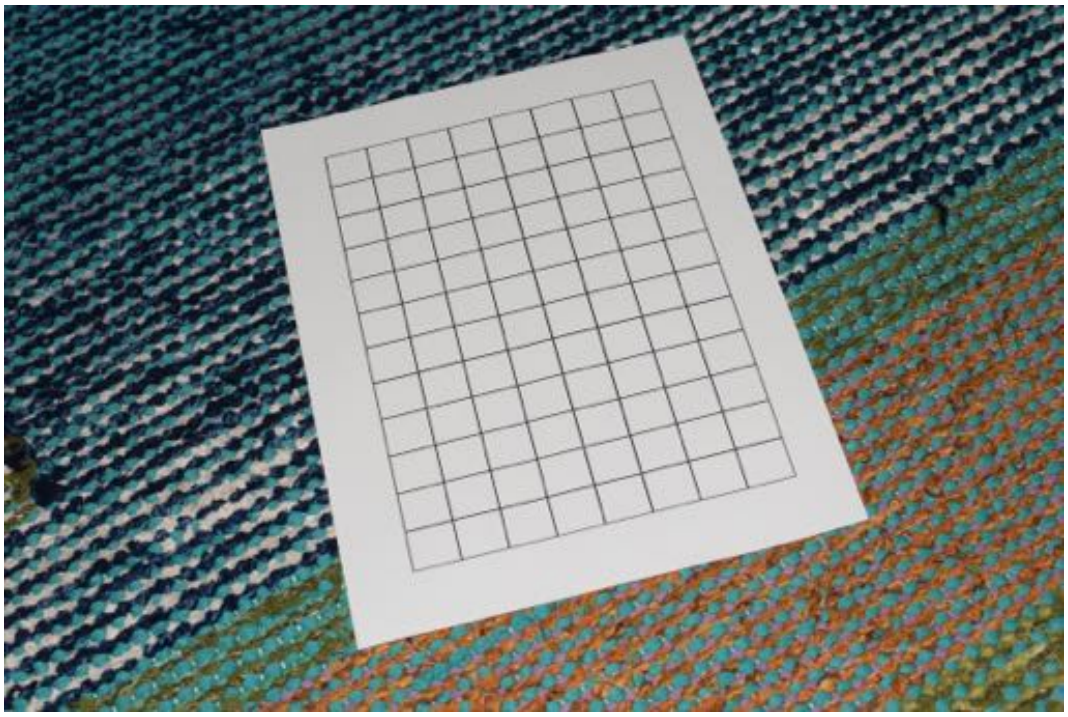


(a) Eingabebild

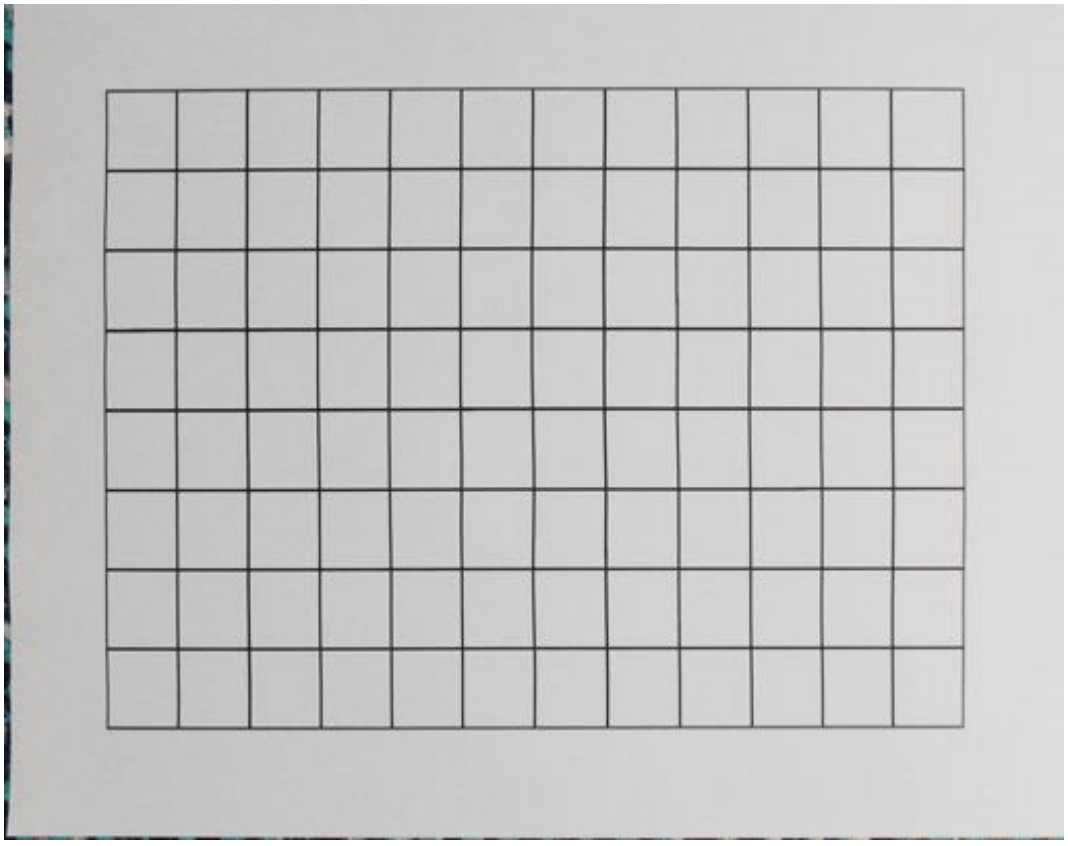


(b) Ausgabebild

Abbildung 12: Texdokument (stark verzerrt)



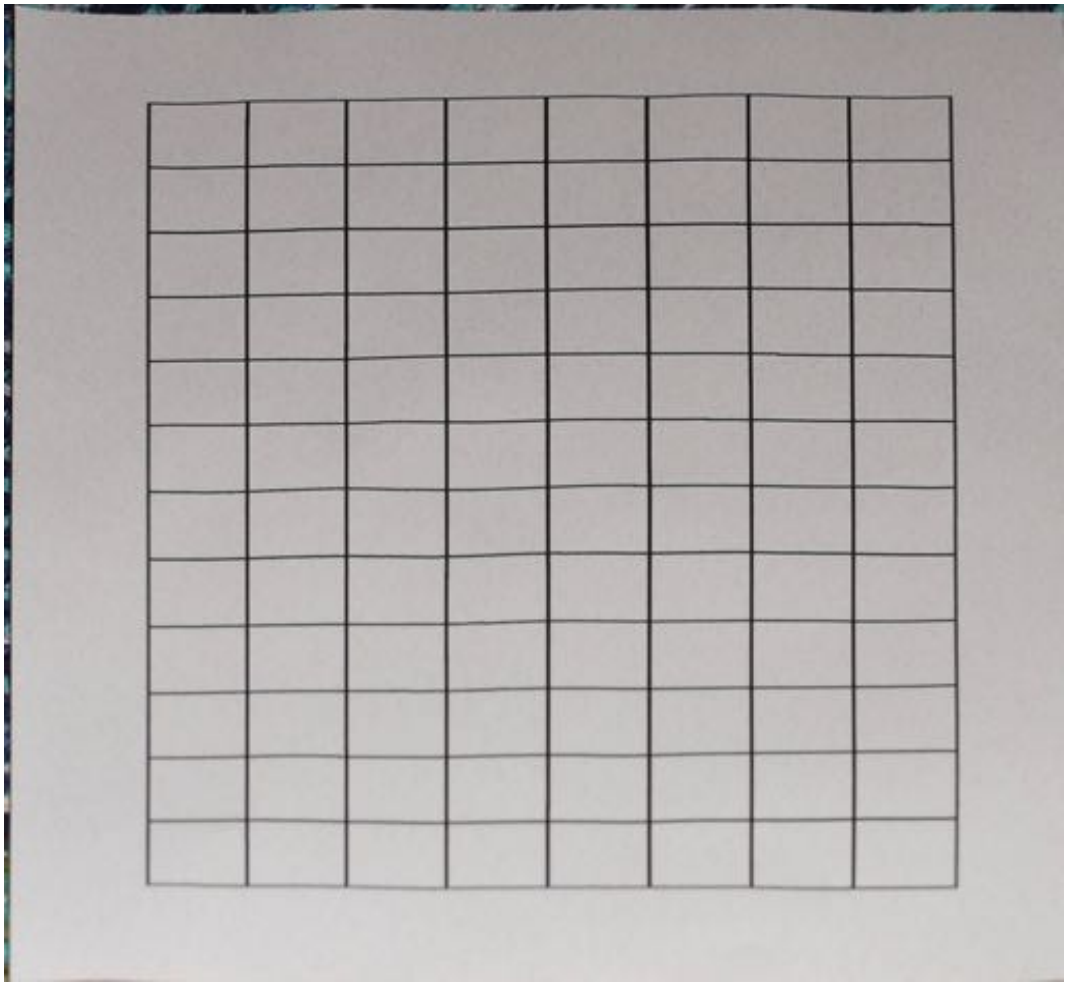
(a) Eingabebild



(b) Ausgabebild

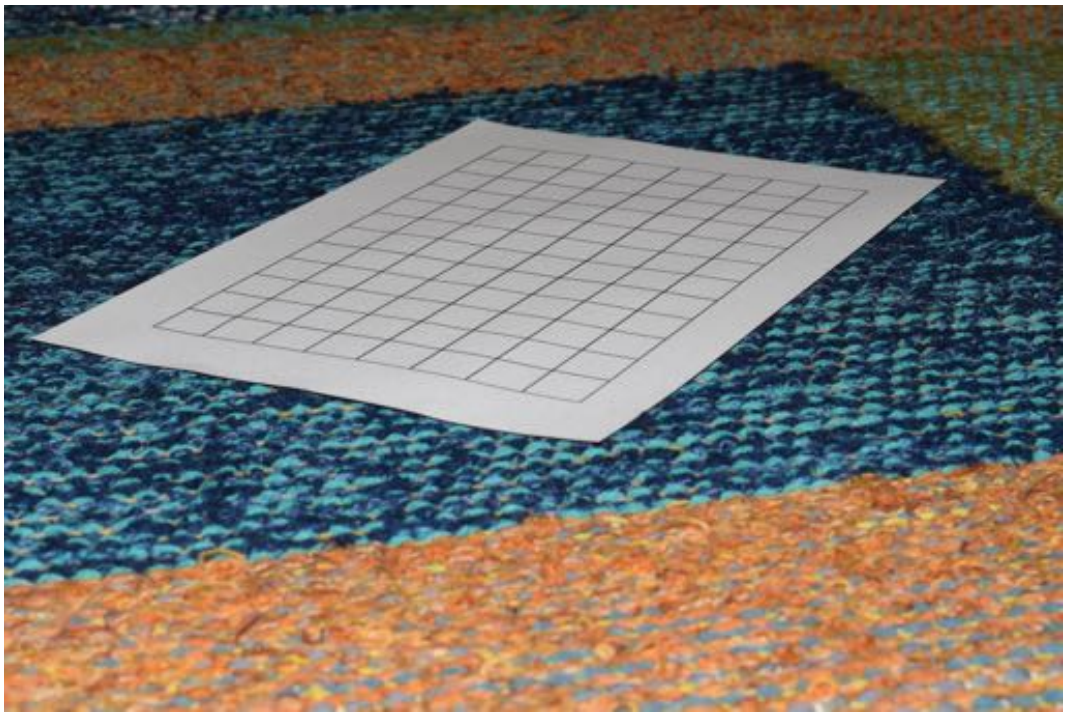
Abbildung 13: Gitter (leicht verzerrt)

(a) Eingabebild

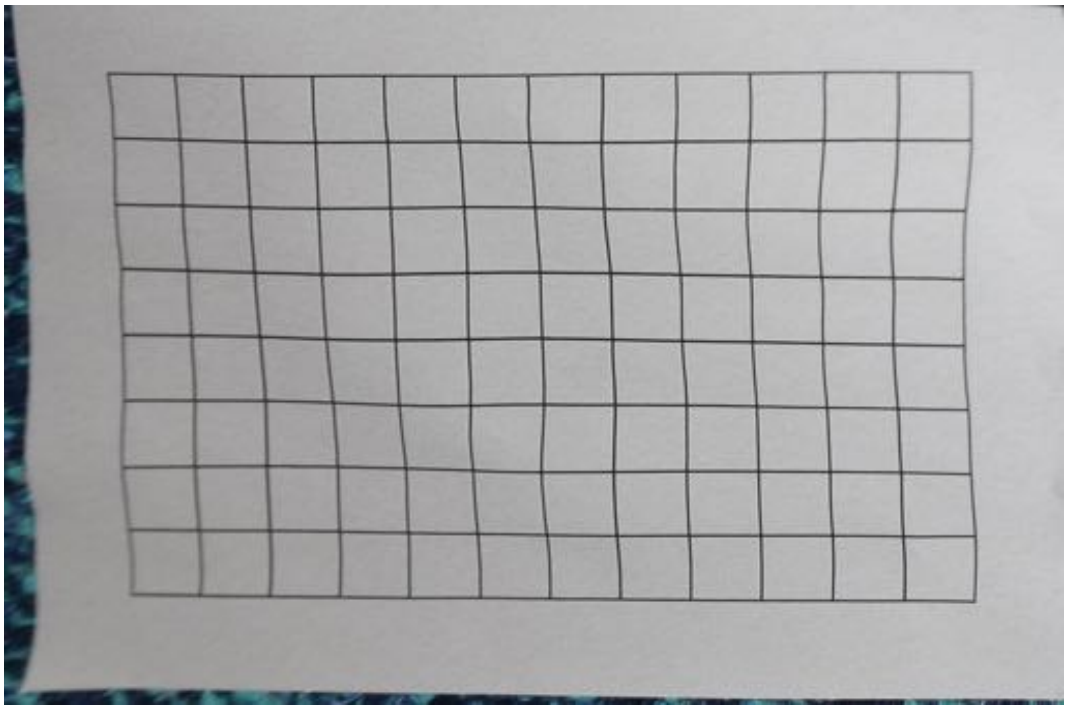


(b) Ausgabebild

Abbildung 14: Gitter (mäßig verzerrt)

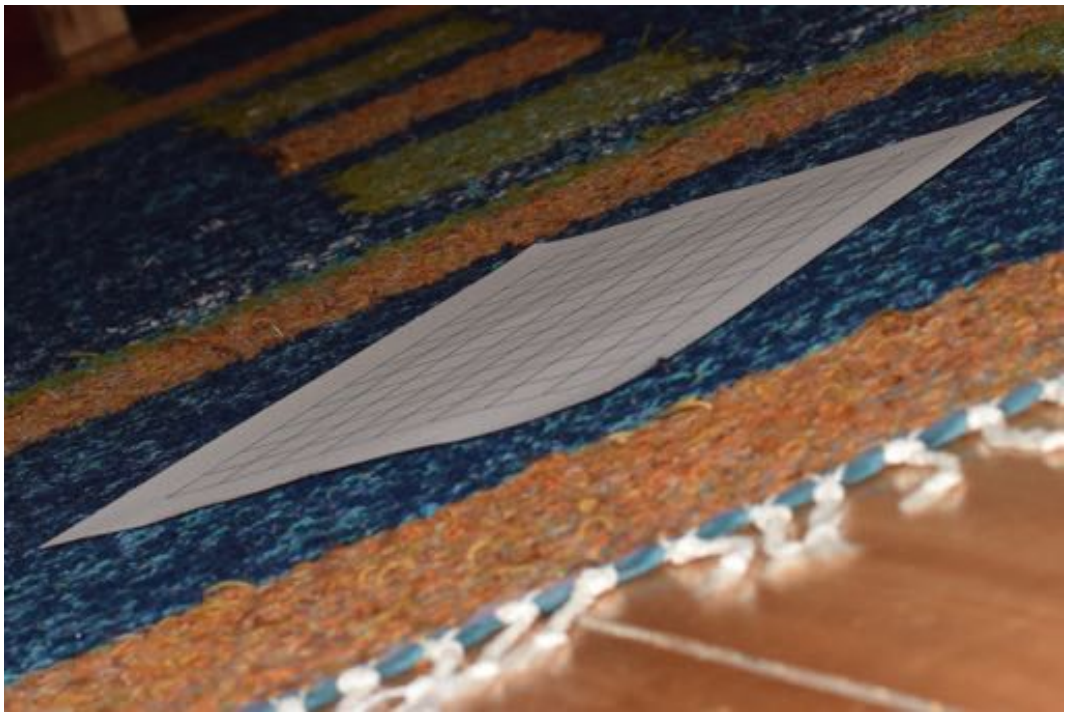


(a) Eingabebild

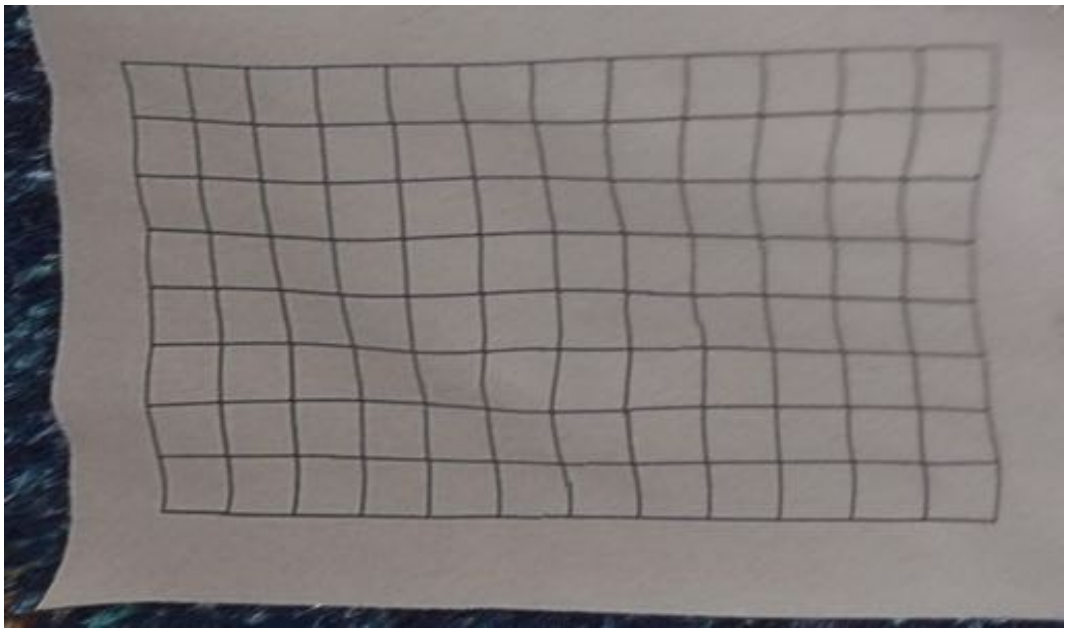


(b) Ausgabebild

Abbildung 15: Gitter (stark verzerrt)



(a) Eingabebild



(b) Ausgabebild

Abbildung 16: Gitter (sehr stark verzerrt)

6.8 Versicherung der selbstständigen Erarbeitung und Anfertigung der Facharbeit

Hiermit versichere ich, dass ich die Arbeit selbstständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt und die Stellen der Facharbeit, die im Wortlaut oder im wesentlichen Inhalt aus anderen Werken (auch aus dem Internet) entnommen wurden, mit genauer Quellenangabe kenntlich gemacht habe.

Göttingen, den 20. März 2016

Hannah Schlüter

6.9 Einverständniserklärung zur Veröffentlichung

Hiermit erkläre ich, dass ich damit einverstanden bin, wenn die von mir verfasste Facharbeit der schulinternen Öffentlichkeit zugänglich gemacht wird.

Göttingen, den 20. März 2016

Hannah Schlüter