PARIS LODRON UNIVERSITÄT SALZBURG

Automatisierte Härteprüfung mit Methoden der Bildverarbeitung und der Computer Vision

Härteprüfung nach Brinell

Christian Arrer

25.04.2011

Eidesstattliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst habe, andere als die angegebenen Quellen nicht verwende habe und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Salzburg, am 22.05.11

Abstrakt

Ziel dieser Arbeit ist es ein Verfahren/ein Softwareprogramm zu entwickeln, welches die Durchmesser und den Ort von kreisrunden oder elliptischen Härteabdrücken in digitalen Bildern bestimmt. Die Software sollte Teil eines Härteprüfverfahrens sein. Dabei wird eine Sintermetallkugel in ein Material gedrückt und die Härte durch das Verhältnis von Eindruckfläche zu Eindruckkraft ermittelt. Als sehr erfolgreich hat sich die Verwendung des Daugman-Integro-Differential-Operators und des Template Matchings herausgestellt.

Abstract

The goal of this work ist the development of a software, that detects the location and the radius of circular or slightly elliptic hardness testing impressions in material surfaces. The software should be part of a hardness testing system. Within this hardness testing machine a sphere of sintered metal is pressed in the material's surface. Then the hardness value is calulated by the quotient of force and imprint surface area. The use of template matching inside the edge image and the use of Daugman's integro differential operator have shown up as very successfull.

Inhaltsverzeichnis

1. Einleitung

1.1 Härteprüfung

Unter Materialhärte versteht man den Widerstand, den ein Festkörper einer Kompression an einer bestimmten Stelle entgegensetzt.

Das Merkmal hart wird in der Technik zur Beschreibung unterschiedlicher Werkstoffeigenschaften benutzt. Allgemein wird der Werkstoffwiderstand gegen das Eindringen eines Fremdkörpers in den oberflächennahen Werkstoffbereich als Härte bezeichnet. Die Härte selbst wird nicht direkt gemessen, sondern aus primären Messgrößen wie Eindruckdurchmesser, Eindringtiefe und Prüfkraft ermittelt.

Da ein mechanischer Zusammenhang zwischen Härte und Festigkeit besteht, ist es in bestimmten Fällen möglich einen Zugversuch durch einfacher durchzuführende Härteprüfungen zu substituieren.

Viele Bauteile wie Achsen, Räder, Turbinenblätter und Wellen unterliegen einer großen Verschleißbeanspruchung. Bei diesen Bauteilen stellt die Härte die Verschleißbeständigkeit dar.

Grundsätzlich werden Bauteile meistens mit einer harten Oberflächenschicht und einer weicheren Innenschicht ausgelegt. In Folge dessen wird eine Unempfindlichkeit gegen Anrisse im Oberflächenbereich und eine hohe Beständigkeit des Werkstoffes erreicht. Äußere mechanische Kräfte werden durch den weichen Kern aufgenommen.

Die wichtigsten Härteprüfverfahren sind:

- Härteprüfung nach Brinell
- Härteprüfung nach Rockwell
- Härteprüfung nach Vickers

Das Härteprüfverfahren nach Brinell (DIN EN ISO 65061) wurde vom schwedischen Ingenieur Johan August Brinell entwickelt. Das Verfahren wird für mittelharte Werkstücke verwendet, als Beispiele seien hier unlegierte Baustähle, sowie Aluminiumlegierungen erwähnt. Eine Hartmetallkugel, als Sintermetall vorgeschrieben, wird mit einer definierten Prüfkraft F in die Oberfläche des Prüfkörpers gedrückt. Die Krafteinwirkung währt dabei nur eine kurze Zeitspanne von 10 bis 15 Sekunden (*Abbildung 1*).

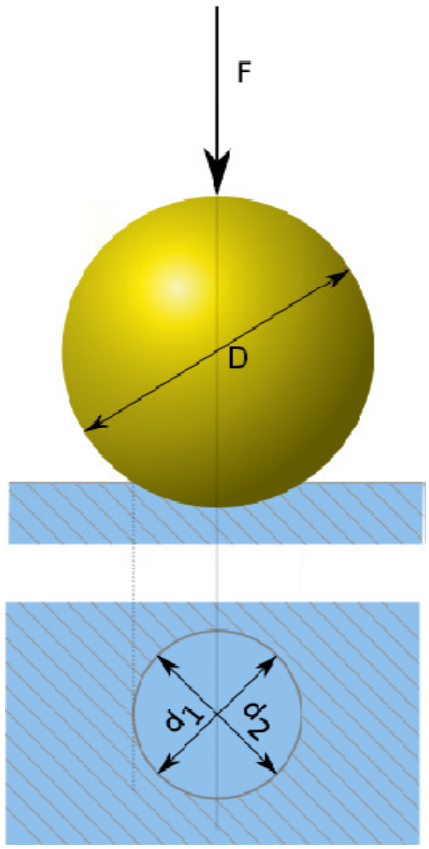


Abbildung 1: Prinzip des Brinell-Härteprüfung mit Eindruckkörper und Prüfeindruck.

Die Brinellhärte wird als Verhältnis zwischen der aufgebrachten Prüfkraft und der Eindruck-Oberfläche des kalottenförmigen Eindrucks definiert. Wird das Brinellverfahren manuell durchgeführt und ausgewertet, so können Anwendereinflüsse nicht ausgeschlossen werden und es ergibt sich eine eingeschränkte Reproduzierbarkeit.

Um die Reproduzierbarkeit der Prüfergebnisse zu erhöhen und Anwendereinflüsse zu minimieren ist es entscheidend, die Prüfabläufe zu automatisieren. Daraus ergibt sich eine erhebliche Zeitersparnis und Kostenreduktion für die Werkstoffprüfung. Die automatisierten Prüfmessgeräte sind bereits mit automatischer Lastaufbringung und geschlossenem Regelkreis für die Laststeuerung ausgestattet. Die Geräte verfügen über einen Revolverkopf zum schnellen Prüfstempelwechsel, sowie eine Digitalkamera die es ermöglicht den Abdruck durch ein Mikroskop aufzunehmen, das Bild zu digitalisieren und computerunterstützt weiter zu verarbeiten.

Im Bereich der digitalen Weiterverarbeitung knüpft diese Arbeit nun an.

1.2 Methodik

Die digitalen Bilder des Prüfeindrucks werden in den Computer übertragen. Matlab-Programme werten diese Bilder anschließend aus. Die Software Matlab bietet Methoden für diese Arbeit. (1)

Die Programme basieren auf Methoden der Mathematik und der Informatik. Es werden zwei Ansätze zur Kreissuche programmiert. Beide Ansätze sind sogenannte brute-force-Verfahren (2). Zum einen wird Template-Matching und zum anderen der Daugman-Integrodifferential-Operator verwendet. Der Daugman-Integrodifferential-Operator ist im Prinzip kein brute-force-Verfahren wird hier aber so eingesetzt. (3)

In beiden Programmen wird der Bresenham-Algorithmus zur effizienten Kreiszeichnung verwendet. (4)

Innerhalb des Template-Matching-Ansatzes kommt der Canny-Kantenerkennungs-Algorithmus zum Einsatz. (5)

In beiden Ansätzen wird CLAHE (Contrast-limited adaptive histogram equalization) bzw. Histogramm-Equalisierung zur Vorverarbeitung verwendet. (6) Um gewisse Bilder zunächst von Bearbeitungsspuren zu befreien wird die Fourier-Transformation und speziell die Fast-Fourier-Transformation verwendet. (7)

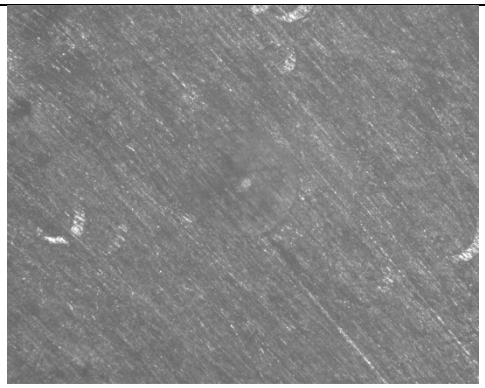
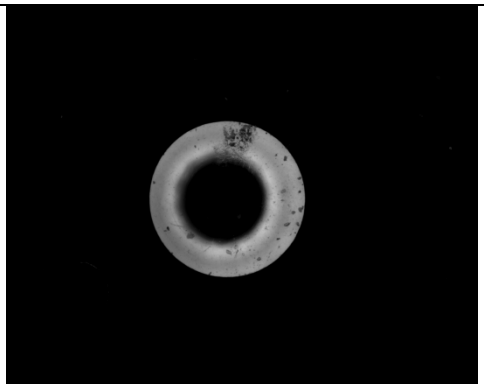
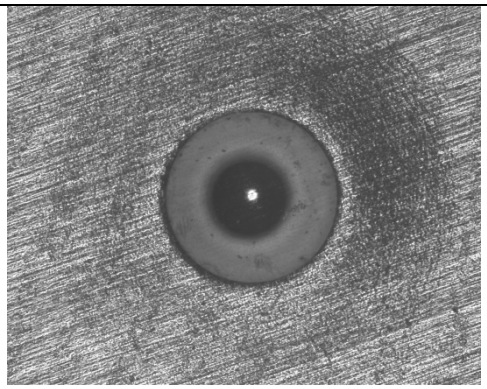
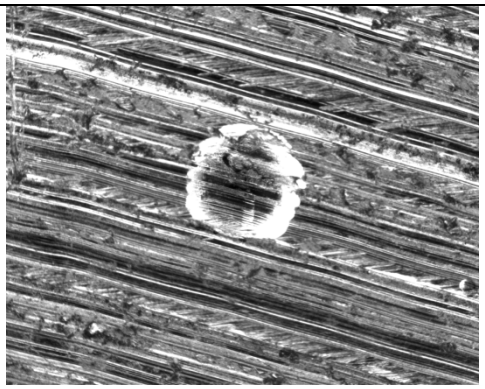
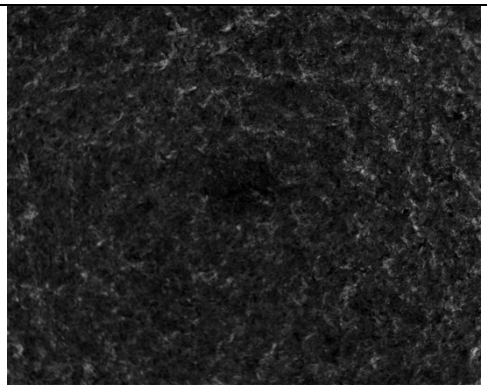
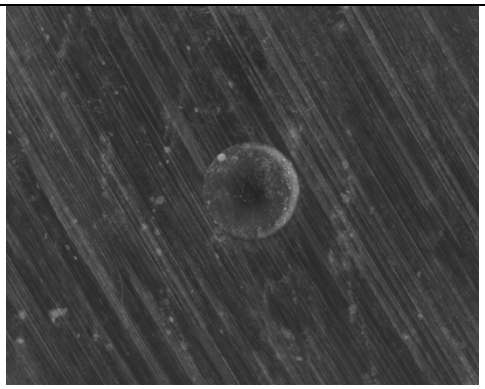
1.3 Aussage

Mit den zwei Hauptprogrammen, die implementiert werden sind zum größten Teil gute Ergebnisse erreichbar. Das größte Problem liegt in der Vorverarbeitung. Es gibt keine sinnvolle Möglichkeit die Parameter des Canny-Edge-Detectors angepasst zu variieren. Des Weiteren wurde keine stabile Entscheidungsfindung für die Durchführung der kontrastadaptiven bzw. der normalen Histogramm-Equalisierung gefunden. Das bedeutet, die Histogramm-Equalisierung verbessert manche Bilder und verschlechtert andere.

2 zu verarbeitende Brinell-Bilder

Die Programme werden mit insgesamt 109 Beispielbildern getestet. Die wichtigsten Vertreter der Bilder werden nun angeführt. Die brisantesten Bilder sind diejenigen, in denen schwarze Bereiche des Hintergrundes in den schwarzen Abdruckbereich einfließen, sowie Bilder die konzentrische Kreise aufweisen, aber auch diejenigen bei denen sich die Schliffspuren in den Abdruck hinein fortsetzten. Am schwersten, oder gar nicht detektiert

werden können Bilder bei denen der Kontrast zwischen Abdruck und Hintergrund extrem gering ist und zusätzlich viel dunklere oder heller Flecken auftreten. Zusätzlich können Bilder mit in höchstem Maße entstellter Kreisform auftreten. Leicht gefunden werden Bilder bei denen der Kontrast zwischen Abdruck und Umgebung sehr hoch ist, bzw. der Abdruck schwarz ist und die Umgebung irgendein Muster hat.

	
Dieser Abdruck hat fast keinen Kontrast zur Umgebung.	In diesm Bild treten konzentrische Kreise auf.
	
Dieses Abdruckbild hat drei konzentrische Kreise.	Die Kreisform dieses Bildes ist stark gestört.
	
Hier liegt ein Bild vor innerhalb dessen fast keine Kreisform auszumachen ist.	In obiger Abbildung hat der Abdruck zwar rechts einen guten Kontrast zur Umgebung, links aber nicht.

3 Recherche in Sammlungen bestehender Implementierungen von Bildsegmentierungsverfahren

Im Rahmen dieser Arbeit wird eine Suche nach Bildsegmentierungsprogrammen im Internet gestartet. Auf *MatlabCentral* werden über 16 Programme gefunden, des Weiteren wird die *OpenCV*-Bibliothek untersucht.

Von den analysierten Segmentierungsverfahren werden hier drei Exemplare vorgestellt und genauer beschrieben.

3.1 Texture Segmentation using GaborFilters and K-means Clustering

Es handelt sich um eine Matlab Implementierung von Natoshi Seo. Das Progamm ist innerhalb eines Kursprojekts an einer Universität entstanden. Durch das Auftreten der verschiedenartig texturierten Metalloberflächen ist dieses Programm durch Suche in diese Richtung entdeckt werden. Gabor-Wavelets können auf Raumfrequenzen von Texturen abgestimmt werden und somit kann der Ort und die Ortsfrequenz festgemacht werden. (8) (9) (10) (11)

3.1.1 Algorithmus

Der Algorithmus steht in Verbindung mit den frühen Stufen der menschlichen Texturwahrnehmung und ist grob an diese angelehnt. Das Programm läuft in 3 Schritten ab.

1. Das Eingabebild wird mit verschiedenen Gaborfiltern in Antwortbilder zerlegt. (*Gaborwavletfilter*)
2. Die Features der Textur werden extrahiert. (*Feature Extraction*)
3. Die Texturpixel können mit verschieden Clusteralgorithmen gruppiert werden. (*K-means Clustering*)

Alle Programmstufen werden nachfolgend nochmals genauer erklärt.

3.1.1.1 Gaborwaveletfilter

Ein Gaborfilter besteht aus einer elliptischen Gauss‘schen Glockenkurve, auf die ein Sinus/Cosinus auf moduliert wird. In der Raumdomäne ist ein Gaborfilter durch folgende Formel beschrieben (*Gl. 1*).

$g(\gamma, \lambda, \psi, \sigma, \theta, x, y) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x'}{\lambda} + \psi\right)$	**Gl. 1**

$x' = x\cos(\theta) + y\sin(\theta)$ $y' = y\cos(\theta) - x\sin(\theta)$ $\lambda \dots Wellenlänge\ des\ Kosinus$ $\theta \dots Ausrichtung\ der\ Normalen\ zu\ den\ Kosiunstalrichtungen$ $\psi \dots Phasenverschiebung\ in\ Grad$ $\sigma \dots Standardabweichung\ der\ Gaussfunktion$ $\gamma \dots lineare\ Exzentrizität\ der\ Gaussfunktion$	

Um eine Halb-Antwortbandbreite von einer oder mehreren Oktaven (Frequenzverhältnis 1:2) sicherzustellen wird der Parameter σ an die Wellenlänge λ angepasst. Die Anpassung ist der Erkennungsbandbreite von einfachen Zellen für hemmende und anregende Streifenzonen nachempfunden. Die Formel hierfür lautet wie folgt (*Gl. 2*).

$\sigma = \frac{1}{\pi}\sqrt{\frac{ln2}{2}}\frac{2^b + 1}{2^b - 1}\lambda$	**Gl. 2**

Die Ausrichtung des Gaborfilters wird, wie in (12) vorgeschlagen, auf die Winkeln $0^\circ, 30^\circ, 60^\circ, 90^\circ, 120^\circ$ und 150° gestellt. Die Frequenzen $f = \frac{1}{\lambda}$ durchlaufen folgende, wie in (13) angegebenen, Werte (*Gl. 3*).

$f_L(i) = 0.25 - \frac{2^{i-0.25}}{n_c}$ $f_H(i) = 0.25 + \frac{2^{i-0.25}}{n_c}$ $i = 1,2,\dots, log_2\left(\frac{n_c}{8}\right)$ $n_c \dots Bildhohe, sollte\ eine\ Zweierpotenz\ sein$	**Gl. 3**

3.1.1.2 Feature Extraction

Die Filter erzeugen Ergebnissbilder die noch nachbearbeitet werden müssen.

Die Werte werden zuerst durch den Tangens Hyperbolicus gesättigt und dann mittels eines Gaussfilters geglättet. Die Standardabweichung des Gaussfilters wird auf $\sigma = 3 * \sigma_s$ gesetzt, wobei σ_s der Größenparameter des Gaborfilters ist.

3.1.1.3 K-means Clustering

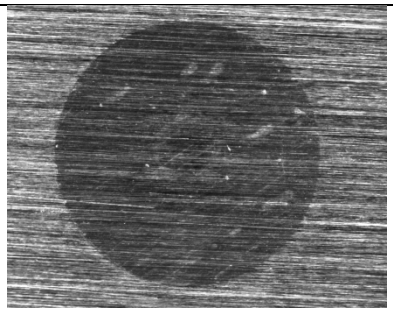
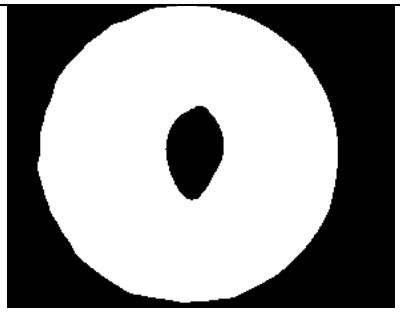
Es könnten verschiedene andere Clustering-Algorithmen zum Einsatz kommen, hier wird der k-means Algorithmus verwendet. Der k-means Algorithmus läuft wie folgt ab:

1. Die Zentroiden der Cluster werden per Zufall initialisiert.
2. Alle Datenpunkte werden zu am nächsten liegenden Zentroiden gruppiert. (euklidische Metrik)
3. Die Zentroiden werden durch die so entstandenen Gruppen neu berechnet.
4. Wenn die Zentroiden sich nicht mehr ändern, dann endet der Algorithmus, ansonsten fängt er wieder mit Schritt 2 an.

Im Clustering-Prozess werden auch die Raumkoordinaten neben den Textureigenschaften als zusätzliche Features inkludiert um dem räumlichen Gebietszusammenhang der Textur Rechnung zu tragen. Das Verfahren liefert bei gleichen Bildern andere Ergebnisse, da die Zentroiden per Zufall initialisiert werden.

3.1.2 Testergebnisse

Um das Programm zu testen müssen die Bilder auf eine Höhe von 256 verkleinert werden, da ansonsten die Berechnung zu lange dauert. Die Segmentierung funktioniert bei Bildern, in denen sich der Abdruck Textur-mäßig gut von der Metalloberfläche unterscheidet, gut. In Bildern in denen die Texturen beider Bereiche fast identisch sind kommt es zu falschen Ergebnissen. Wie vorher schon erwähnt, kann sich die Segmentierung desselben Bildes durch die zufällige Initialisierung der Cluster bei erneuter Bearbeitung ändern. Das Problem liegt auch bei der Variation der Filterparameter. Tatsächlich handelt sich um ein allgemeines Bildsegmentierungsprogramm, in dem die Kreisform nicht berücksichtigt wird. Die Kreisform ist aber die am besten vorhandene Eigenschaft in den Testbildern. Der Algorithmus hat auch Probleme bei den Texturgrenzen. Nachfolgende *Tabelle 1* zeigt ein gutes und ein schlechtes Ergebnis.

erfolgreiche Segmentierung	
	

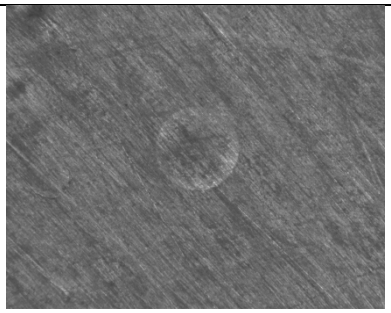
gescheiterte Segmentierung	
	

Tabelle 1

Eine Möglichkeit dieses Verfahren zu verwenden liegt in einer Kombination mit Kanteninformationen. Das Verfahren könnte möglicherweise verwendet werden um mit dem Daugman-Integro-Differential-Operator gefundene Kreise auf Plausibilität zu prüfen.

3.2 Region-Growing

Es handelt sich bei diesem Programm um eine Matlab Implementierung von Dirk Jan Kroon von der Universität Twente. Das Programm lässt einen Bereich ausgehend von einem Punkt wachsen. Die Funktionsweise des Programms sowie die Ergebnisse der Anwendung auf die Brinellbilder wird nun kurz dargestellt.

3.2.1 Algorithmus

Das Programm startet mit einem Seedpoint, dabei handelt es sich um irgendeinen vorgegebenen Startwert des Bereichswachstums. Anschließend werden die Nachbarn der Region bestimmt. Derjenige Nachbarpixel mit der geringsten Abweichung zum Mittelwert der Region-Helligkeit wird der Region einverleibt. Sobald sich der Bereich vergrößert hat muss der Helligkeitsmittelwert neu berechnet werden. Falls alle Randpixel über einen gewissen Schwellwert vom Regionsmittelwert der Helligkeit abweichen bricht das Wachstum ab. Das Programm endet auch dann, wenn das ganze Bild ausgefüllt worden ist.

Die Software benötigt beim Start die Koordinaten des ersten Seedpoints.

3.2.2 Testergebnisse

Für manche Bilder funktioniert das Verfahren gut. Das Problem ist unteranderem die Wahl des Startpixels. Am besten wäre es den Start im Abdruck zu wählen, was aber ohne vorherige Einschränkung des Abdrucks nicht möglich ist. Das Programm könnte auch dahingehend verändert werden, dass der gesamte Bildrand als Startbereich verwendet wird. Ein weiteres Problem ist die automatische Wahl des Abbruchschwellwerts. Bei großen

Bildern ist das Verfahren sehr langsam. Nachfolgend werden nochmals Segmentierungsergebnisse gegenübergestellt (*Tabelle 2*).

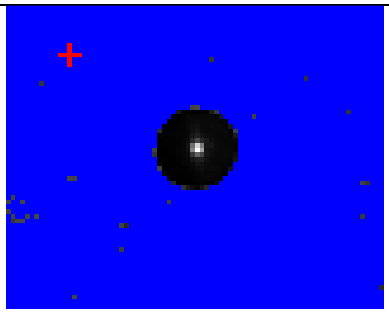
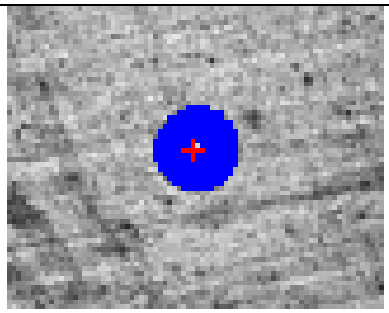
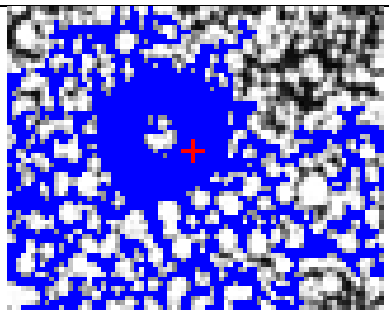
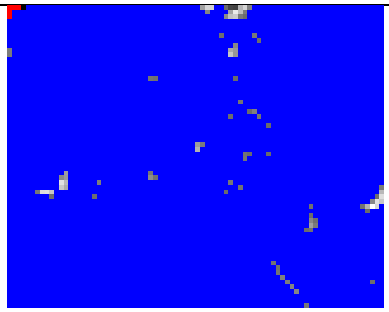
erfolgreiche Segmentierung	
	In Bildern in denen sich der Abdruck gut von der Oberfläche unterscheidet, lässt sich ein gutes Segmentierungsergebnis erzielen. Das rote Kreuz markiert den Starpixel. Das Problem liegt bei der Wahl des Abbruchschwellwerts.
	Am besten ist es den Abdruck zu treffen. Wenn sich dieser Bereich gut von der Umgebung abhebt und einigermaßen gleichmäßig ist, dann lässt sich eine gute Segmentierung erreichen. Die Segmentierung ist besser als wenn der Seedpoint in der Umgebung liegt.
gescheiterte Segmentierung	
	In Bildern in denen der schwarze Abdruckberiech mit dem Hintergrund zusammenfließt, schlägt das Verfahren fehl. Wie weit die Region in die Umgebung ausrinnt hängt vom gewählten Abbruchschwellwert ab.
	Das Verfahren funktioniert auch bei Bildern mit schlechtem Abdruck Hintergrund kontrast nicht. Gerade bei solche Aufnahmen macht es keinen Unterschied, ob das Verfahren im Abdruck oder in der Umgebung startet.

Tabelle 2: bessere und schlechtere Segmentierungsergebnisse

3.3 Statistical Region Merging

Die genaue Funktionalität des Programmes kann nicht erfasst werden. Die Beschreibung der Autoren Richard Nock und Frank Nielsen wird kurz zusammengefasst.

3.3.1 Algorithmus

Sehen ist ein Ableitungsproblem, es ist die Frage was die beobachteten Daten erzeugt hat. Eine Sammlung von Pixel wird in eine Partition vom Objektbereichen gruppiert. Um die Lösung der Fragestellung anzugehen ist die Umwandlung in eine wohldefinierte Problemstellung nötig. (14)

Das betrachtete Bild wird als durch ein ideal segmentiertes Bild erzeugt angenommen. Im idealen Bild haben alle Pixel einer Gruppe denselben Erwartungswert in allen Farbkanälen. (14)

Das Programm basiert unteranderem auch auf den folgenden Sätzen.

*„**Theorem 1** (The independent bounded difference inequality). Let* $\boldsymbol{X} = (X_1, X_2, \dots, X_n)$ *be a family of* n *independent r.v. with* X_k *taking values in a set* A_k *for each* k*. Suppose that the real-valued function* f *defined on* $\prod_k A_k$ *satisfies* $|f(x) - f(x')| \leq c_k$ *whenever vectors* x *and* x' *differ only in the* k*th coordinate. Let* μ *be the expected value of the r.v.* $f(\boldsymbol{X})$ *Then, for any* $\tau \geq 0$*,*

$Pr(f(\boldsymbol{X}) - \mu \geq \tau) \leq exp(-2\tau^2 / \sum_k (c_k)^2)$*“* (14)

*„**Theorem 2** With* $probability \geq 1 - 0(|I|\delta)$*, the segmentation on I satisfying* A *is an overmerging of* I^**, that is:* $\forall O \in s^*(I), \exists R \in s(I): O \subseteq R$*“* (14)

Es folgt die Beschreibung des Algorithmus.

*„**Our Algorithm: SRM** In 4-connexity, there are* $N < 2|I|$ *couples of adjacent pixels. Let* S_I *be the set of these couples. Let* $f(p, p')$ *be a real valued function, with* p *and* p' *pixels of I. Our segmentation algorithm, SRM (for Statistical Region Merging) is simple. We first sort the couples of* S_I *in increasing order of* $f(.,.)$*, and then traverse this order only once. We make for any current couple of pixels* $(p, p') \in S_I$ *for which* $R(p) \neq R(p')$ *(*$R(p)$ *stands for the current region to which p belongs) the test* $P\big(R(p), R(p')\big)$*, and merge* $R(p)$ *and* $R(p')$ *if it returns true. The objective is obviously to choose* $f(.,.)$*, so to approximate A as best as possible.“* (14)

3.3.2 Testergebnisse

Da es nicht möglich ist die genaue Funktionalität des Programms zu erfassen, macht es keinen Sinn die vorgegebenen Q-Levels zu verstellen.

Das Verfahren sieht sehr vielversprechend aus und könnte vielleicht auch mit dem Daugman-Integro-Differential-Operator kombiniert werden. Um das Verfahren besser einsetzen zu können muss Aufwand investiert werden um es zu verstehen. Grundsätzlich ist anzumerken, dass das Programm verschiedene Segmentierungsstufen erzeugt (*Tabelle 3*).

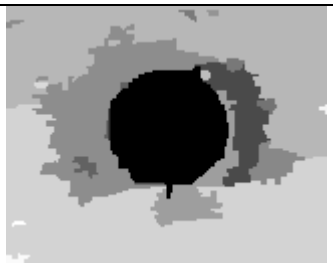
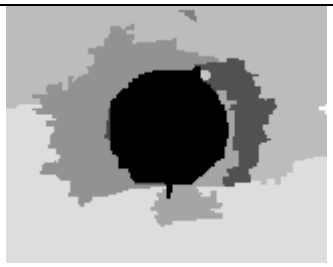
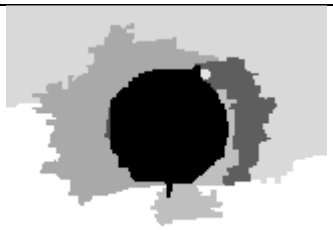
Segmentierungstufen	
	
	
	
	

Tabelle 3: Segmentirungsstufen - in den letzten Stufen wird tatsächlich nur ein Bereich gefunden

4 Zusammenfassung in den implementierten Programmen verwendeten Algorithmen

4.1 Kantenerkennung

Um die verwendeten Kantenerkennungs-Algorithmen zu beschreiben ist zunächst eine Erklärung des Begriffs der Kanteneigenschaften (*edge properties*) nötig.

Anschließend werden der Canny-Kantenerkennungs-Algorithmus (*canny edge detector*) und als Teil davon der Sobel-Operator beschrieben.

4.1.1 Kanteneigenschaften (*edge properties*)

Als Beispiel wird ein Graustufenbild des halbbeleuchteten Erdmondes herangezogen. Die Helligkeit jedes Bildpunktes wird mit $2^8 = 256$ verschiedenen Abstufungen dargestellt. Somit ist für jeden Pixel ein Rohspeicherplatzbedarf von einem Byte erforderlich. Der Übergang von beleuchteter Mondoberfläche zum Weltall soll jetzt genauer betrachtet werden (*Abbildung 1*).

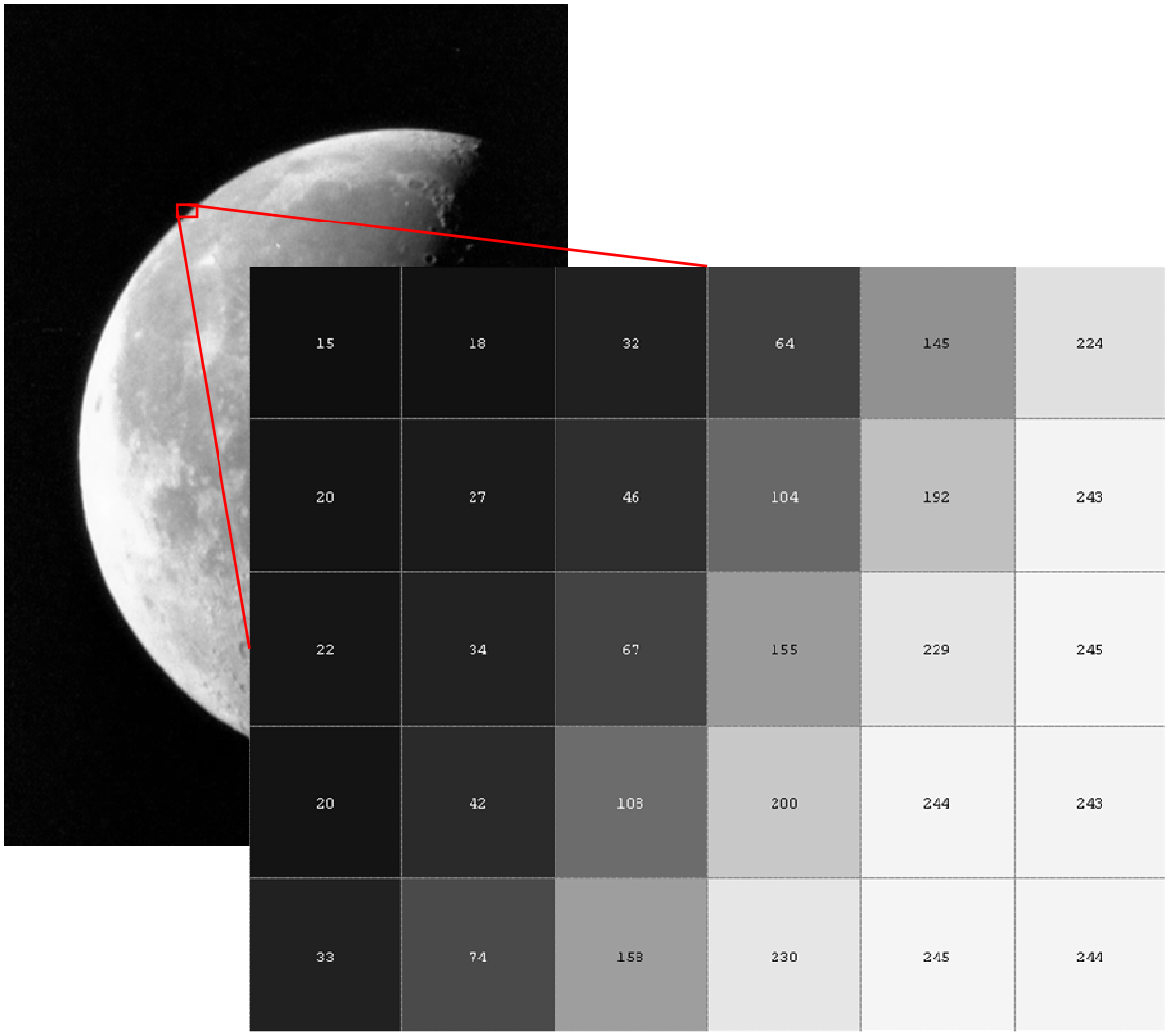


Abbildung 1: Vergrößerung des Übergangs Mondoberfläche - Weltall (0 ist schwarz, 255 ist weiß)

In *Abbildung 1* sind links oben die Pixel des Weltraums und rechts unten die des Mondes zu erkennen. Zusätzlich kann der Graustufen- bzw. Helligkeitswert abgelesen werden.

Mit optischen Sensoren ausgestattete Lebewesen nehmen den Helligkeits-Unterschied zwischen Mondoberfläche und Weltraum wahr und interpretieren dann aufgrund der Grenzlinie das Objekt Mond und den Hintergrund Weltall. Um eine automatisierte Erkennung von Kanten umzusetzen, müssen deren Eigenschaften genauer betrachtet werden. In der Vergrößerung des Übergangs Mond - Weltraum (*Abbildung 1*) ist eine starke Differenz in den Grauwerten festzustellen. Die Suche von Kanten in einem Graustufenbild ist also eine Suche nach großen Grauwertsprüngen.

Ein Graustufenbild kann als skalare Funktion von Pixelkoordinaten interpretiert werden. Die Stärke der Änderung in den Grauwerten lässt sich mathematisch durch den Gradienten ausdrücken. Der Gradient ist definiert als Vektor der partiellen Ableitungen jeder Variable einer skalaren Funktion. Als Vektor hat der Gradient eine Richtung und einen Betrag. Mit

der Berechnung dieser beiden Werte für jeden Pixel wird der Gradient an jeder Stelle festgelegt. Dazu zeigt *Abbildung 2* eine Vergrößerung eines Pixels und seiner drei mal drei Nachbarschaft (Helligkeitswert 200) aus *Abbildung 1*.

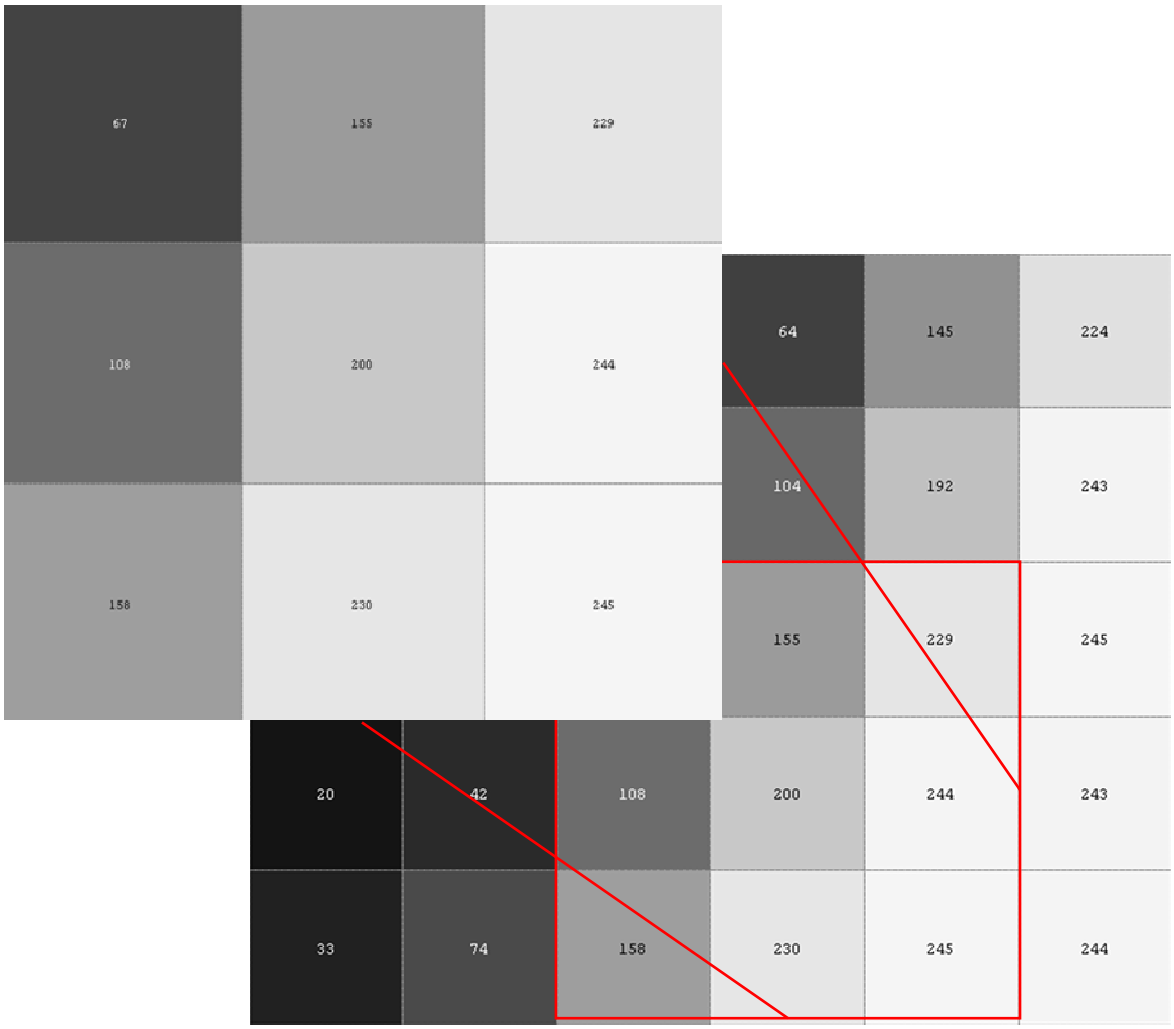


Abbildung 2: Pixel und drei mal drei Nachbarschaft

Das Pixelkoordinatensystem des Bildes wird standardmäßig im linken oberen Eck platziert. Die Indexnummern nehmen auf der vertikalen Achse von oben nach unten und auf der horizontalen von links nach rechts zu (*Abbildung 3*).

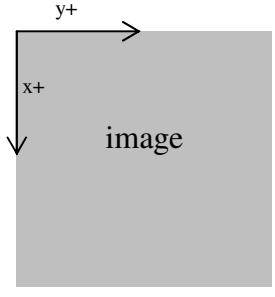


Abbildung 3: Koordinatensystem der Bildmatrix

Die partiellen Ableitungen werden in der Vertikalen (x-Richtung) durch die Grauwertdifferenzen der Punkte oberhalb und unterhalb des jeweiligen Pixels, und in der Horizontalen (y-Richtung) durch die Grauwertdifferenzen der Punkte links und rechts des jeweiligen Pixels angenähert (*Gl. 4*).

$grad\, f\big((x,y)\big) \approx \big(f(x+1) - f(x-1), f(y+1) - f(y-1)\big)$	**Gl. 4**

Für den Pixel des Grauwerts 200 (*Abbildung 2*) ist der Gradientenvektor $(75, 136)$. Der Betrag des Gradientenvektors ist ~ 155 und die Richtung, ausgehend von der vertikalen Achse übergehend in die horizontale, liegt bei $\theta = \sim 61°$ (*Abbildung 4*).

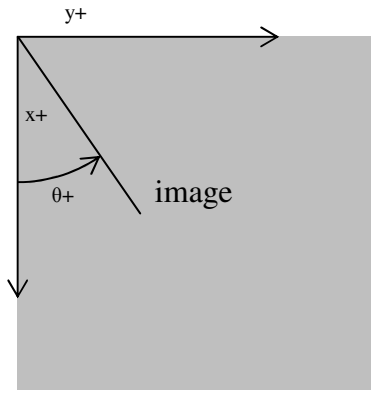


Abbildung 4: Winkelrichtung des Gradienten

Die aus der Berechnung des Gradienten resultierenden Richtungs- und Intensitätsinformationen fließen in die Kanteneigenschaften eines Pixels ein. Die Kanteneigenschaften eines Pixels sind die Kantenstärke und die Kantenrichtung, wobei die Kantenstärke der Betrag des Gradientenvektors ist und die Kantenrichtung senkrecht zum Gradientenvektor steht. Die Kantenstärke des Pixels mit Grauwert 200 (*Abbildung 3*) ist ~ 155 und die Kantenrichtung ist $\varphi = \sim 151°$.

Auch Pixel, die nicht zu einem echten Kantenzug gehören, können hohe Kanteneigenschaften aufweisen. Rauschpixel haben hohe Kanteneigenschaften, sind aber nicht Teil einer echten Kante. Einfache Kantenerkennungsverfahren können dadurch leicht in die Irre geführt werden.

4.1.2 Sobel-Operator

Der Sobel-Operator zählt zu den Kompass-Operatoren und er ist prinzipiell eine drei mal drei Matrix. Je nach Ausrichtung des Operators wird die Komponente des Gradienten in eine der acht Nachbarrichtungen eines Pixels bestimmt. Dabei sind vier verschiedene

Sobel-Matrizen sinnvoll, da entgegengesetzte Richtung durch eine einzige Sobel-Matrix berechnet werden und das Vorzeichen der Gradientenkomponente die Orientierung bestimmt (*Abbildung 5*).

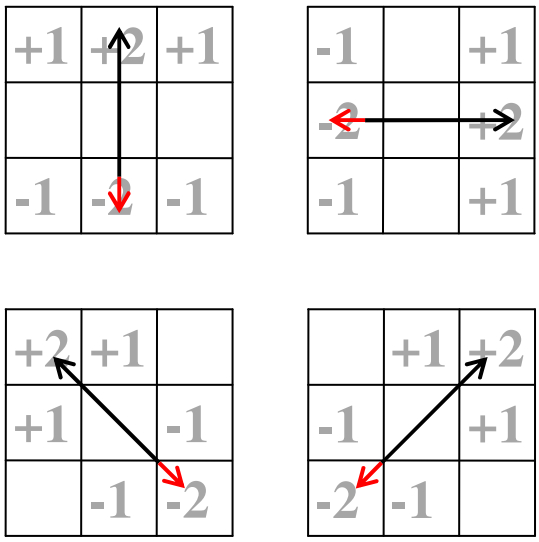


Abbildung 5: positive Gradientenrichtungen

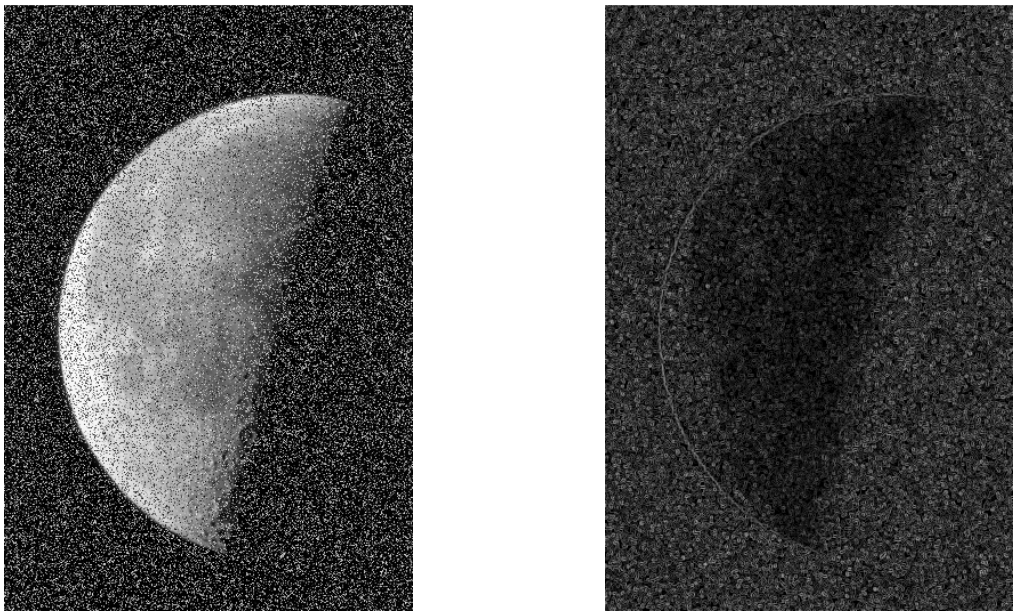
Die Matrizen des Sobel-Operators für Haupt- und Nebenrichtungen haben einfache Gestalt (*Gl. 5*).

$$K_{main} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_{diag} = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix} \quad \textbf{Gl. 5}$$

Werden beide Matrizen um 90° rotiert entstehen insgesamt vier Matrizen für acht Richtungen. Je nach Ausrichtung, werden die in Operator-Richtung gelegenen Nachbarpixel mit den Zahlen -2 und 2 stärker berücksichtigt als die mit den Zahlen 1 und -1 gewichteten. Der Sobel-Operator berechnet den Gradienten und führt gleichzeitig eine Glättung, durch Einbeziehen zur Operator-Richtung diagonalen Nachbarn, aus.

Die Anwendung des Sobel-Operators auf ein Bild ist das Faltungsprodukt zwischen Sobel-Matrix und Bild-Matrix.

Der Sobel-Operator reagiert auf normale Kanten genauso stark wie auf Rauschpixel und wird von diesen sehr leicht in die Irre geführt. Ein verrauschtes Bild muss vor Anwendung des Operators vom Rauschen befreit werden (*denoising*). Nachfolgende *Abbildung 6* zeigt das schlechte Ergebnis des durch Rauschen verwirrten Sobel-Operators.



(a) (b)

Abbildung 6: a: verrauschtes Bild b: Sobel-Operator verwirrt durch Rauschen

Des Weiteren ist der Sobel-Operator nicht im Stande langsame Kantenverläufe zu detektieren. Eine unscharfe Kante ohne schlagartigen Gradientensprung liefert keine Antwort auf Anwendung des Sobel-Operators.

Die Probleme des Sobel-Operators werden durch den Canny-Kantenerkennungs-Algorithmus größtenteils beseitigt.

4.1.3 Canny-Kantenerkennungs-Algorithmus (canny edge detector)

Der Canny-Algorithmus wurde 1986 von John Canny entwickelt. Trotz des Alters ist er ein Standardverfahren der Kantenfindung. Für verrauschte Stufen-Kanten (*step edges*) ist er optimal geeignet.

Der Algorithmus sollte die folgenden, vom Entwickler verlangten, drei Ziele erfüllen:

- **Entdeckung (*detection*):** Die Findung von echten Kanten soll maximiert und die falsche Erkennung von unechten Kanten minimiert werden. Dieses Ziel verlangt die Maximierung des Signal-Rausch-Verhältnisses (*signal to noise ratio*).
- **Ortsgenauigkeit (*localisation*):** Die Abweichung von den tatsächlichen Bildkanten und den vom Algorithmus entdeckten soll minimal sein.
- **eindeutige Antwort (*one response*):** Eine echte Kante darf nicht als mehrere Kanten gefunden werden.

Diese Ziele wurden durch den Entwickler auch mathematisch definiert. Der Canny-Kantenerkennungs-Algorithmus ist fast jeder Software für Bildverarbeitungsprobleme enthalten.

Der Canny-Algorithmus läuft in 5 Schritten ab.

- **Glätten** (*smoothing*): Das Bild wird durch einen Gaussfilter geglättet.
- **Gradienten finden** (*finding gradients*): Pixel mit einem hohen Gradienten bilden Kanten.
- **Unterdrückung von Nicht-Maxima** (*non maxima suppression*): Innerhalb eines Kantenverlaufs werden nur die lokalen Maxima als Kantenpixel behalten.
- **doppelter Schwellwert** (*double thresholding*): Kandidaten für Kanten werden durch zwei Helligkeitswert-Schranken ausgewählt.
- **Kanten Verfolgung durch Pfadabhängigkeit** (*edge tracking by hysteresis*): Entdeckte Kanten, die nicht mit einer starken Kante zusammenhängen werden verworfen.

Die Schritte des Canny-Algorithmus werden nun im Detail erklärt. Die Ergebnisse aller Stufen werden an einem Testbild (*Abbildung 7*) gezeigt.

Abbildung 7: Testbild

4.1.3.1 Glätten

Als Glätten bezeichnet man die Berechnung des Faltungsprodukts der Bildmatrix mit einem Gaussfilter. In jedem Bild ist Rauschen zu erwarten. Eine Glättung unterdrückt die Rauschpixel und verhindert eine große Anzahl an falschen Kantenantworten. Die Matrix eines Gaussfilters kann verschiedene Gestalten haben. Es folgt ein typischer Gaussfilter (*Gl. 6*), der auf *Abbildung 7* angewandt wird:

$A = \frac{1}{159}\begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$	**Gl. 6**

Die Normierung auf die Summe der Elemente in *Gl. 6* wird vom Faltungsprodukt verlangt. Im Falle einer Nicht-Normierung kann der Wertebereich in der Ergebnismatrix überschritten werden. Anwendung des Gaussfilters liefert das Ergebnis *Abbildung 8*.

Abbildung 8: Testbild nach Glättung mit Gaussfilter

4.1.3.2 Gradienten finden

Innerhalb des Canny-Kantenerkennungs-Algorithmus kann der Sobel-Operator auch durch andere Kompass-Operatoren ersetzt werden.

Der Sobel-Operator wird innerhalb des Canny-Kantenerkennungsverfahrens zur Gradienten-Berechnung verwendet. Der Gradient wird nicht einzeln durch alle vier Sobel-Operatoren in jede Richtung ermittelt, sondern nur in beide Hauptrichtungen. Es werden folgende Kern-Matrizen verwendet (*Gl. 7*).

$K_x = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, K_y = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$	**Gl. 7**

Nach Berechnung des Faltungsprodukts zwischen Bild und Sobel-Operator entstehen zwei Gradienten Matrizen, eine mit den Gradientenbeträgen in x-Richtung und eine mit den Gradientenbeträgen in y-Richtung. Um den endgültigen Betrag des Gradienten zu erhalten können die Gradienten quadratisch addiert werden (*Gl. 8*).

$G = \sqrt{{G_x}^2 + {G_y}^2}$	**Gl. 8**

Der Berechnungsaufwand steigt durch iterative Verfahren der Wurzelberechnung und durch den erhöhten Speicherbedarf von Geleitkommadarstellungen. Der Stadtviertel-Abstand (*manhattan distance*) kann anstatt des euklidischen Abstands verwendet werden. Er ist nicht exakt, reduziert aber den Rechenaufwand (*Gl. 9*).

$G = \lvert G_x \rvert + \lvert G_y \rvert$	**Gl. 9**

Der Winkel der Gradientenrichtung kann mit der Umkehrfunktion des Tangens berechnet werden (*Gl. 10*), bei dieser Berechnung sind die Quadranten zu berücksichtigen.

$\theta = \begin{cases} \tan^{-1}\left(\frac{G_x}{G_y}\right) \; wenn \; G_x > 0 \wedge \; G_y > 0 \\ \pi + \tan^{-1}\left(\frac{G_x}{G_y}\right) \; wenn \; G_x < 0 \wedge \; G_y > 0 \\ \pi + \tan^{-1}\left(\frac{G_x}{G_y}\right) \; wenn \; G_x < 0 \wedge \; G_y < 0 \\ 2\pi + \tan^{-1}\left(\frac{G_x}{G_y}\right) \; wenn \; G_x > 0 \wedge \; G_y < 0 \end{cases}$	**Gl. 10**

Der Winkel θ wird zunächst mit großer Genauigkeit gespeichert. *Abbildung 9* zeigt das Graustufenbild des Gradientenbetrags.

Abbildung 9: Gradientenbetrag

4.1.3.3 Unterdrückung von Nicht-Maxima

Aus dem vorherigen Schritt resultieren unscharfe Kantenzüge. Der Nicht-Maximum-Unterdrückungs-Algorithmus (*non maximum surpression algorithm*) wird angewandt um dünne scharfe Kantenverläufe zu erhalten.

Zunächst werden die Gradientenrichtungen auf 45°-Richtungen gerundet. Die Gradienten haben dann nur noch acht diskrete Richtungen.

Jeder Gradientenpixel wird mit den Nachbarpixel, die in seiner Gradientenrichtung liegen vergleichen. Der Pixel wird nur dann behalten, falls er ein lokales Maximum und damit größer als seine Nachbarn ist.

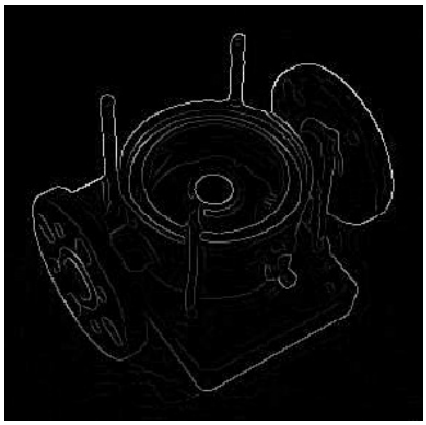


Abbildung 10: Gradientenbild nach Unterdrückung der Nicht-Maxima

4.1.3.4 doppelter Schwellwert

Um die schwachen Kanten aus dem letzten Schritt (*Abbildung 10*) zu unterdrücken könnte ein Grauschwellwert eingesetzt werden. Der Canny-Algorithmus verwendet stattdessen einen doppelten Schwellwert. Die bereits gefundenen Gradienten-Maxima werden mit einem niedrigen (*low threshold*) und einem hohen Schwellwert (*high threshold*) klassifiziert. Alle Gradientenpixel unterhalb des niedrigen Schwellwerts werden verworfen. Pixel zwischen niedrigem und hohem Schwellwert werden als schwach (*weak*) markiert. Liegt der Gradient über dem hohen Schwellwert wird der entsprechende Pixel als stark (*strong*) eingestuft.

Zusammengefasst ergeben sich drei Arten von Pixel:

- **keine Kante:** Pixel deren Gradientenbetrag unter dem niedrigen Schwellwert liegt.
- **schwache Kante:** Pixel deren Gradientenbetrag zwischen hohem und niedrigen Schwellwert liegt.

- **starke Kante:** Pixel deren Gradientenbetrag über dem hohen Schwellwert liegt.

Ein Verwendung der Schwellwerte 20 (niedriger Schwellwert) und 80 (hoher Schwellwert) auf *Abbildung 10* ergibt *Abbildung 11.*

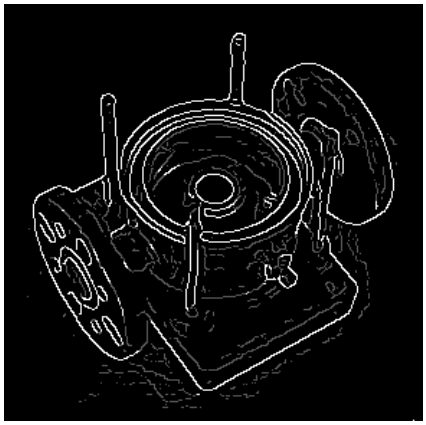


Abbildung 11: doppelter Schwellwert

4.1.3.5 Kanten Verfolgung durch Pfadabhängigkeit

Die schwachen Kanten aus dem vorherigen Schritt werden nachbearbeitet. Schwache Kanten werden nur dann als echte Kanten behalten, falls sie mit einer starken Kante zusammenhängen.

Rauschen produziert mit großer Wahrscheinlichkeit schwache Kanten. Diese schwachen Kanten haben geringere Chancen mit einer echten Kante verbunden zu sein und werden gelöscht. Schwache Kanten, die tatsächlich zu echten Kanten im Bild gehören sind höchstwahrscheinlich mit starken verbunden und werden nicht verworfen.

Die Kantenverfolgung kann mit verschiedenen Methoden umgesetzt werden. Am einfachsten ist es die schwachen Kanten in Kleckse (*blobs*) zu gruppieren. Ein Pixel gehört dann zu einem Kleks, falls sich ein Kleckspixel auf einer seiner acht Nachbarpositionen befindet. Ist in der Nachbarschaft eines Kleckspixels ein starker Kantenpixel, so wird der Klecks als echter Kantenzug behalten.

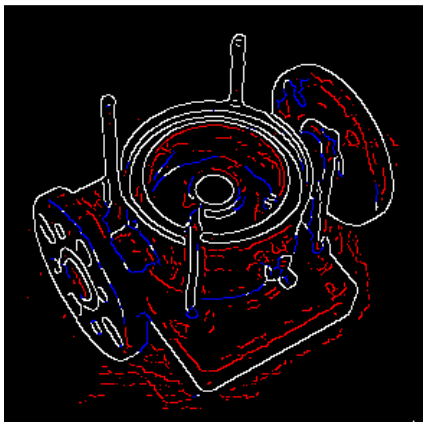


Abbildung 12: Kantenverfolgung

In *Abbildung 12* sind schwachen Kanten, die in Verbindung mit starken stehen, blau, und schwache Kanten, die nicht in Verbindung mit starken Kanten stehen, rot eingefärbt. Die blauen bleiben erhalten, die roten werden verworfen.

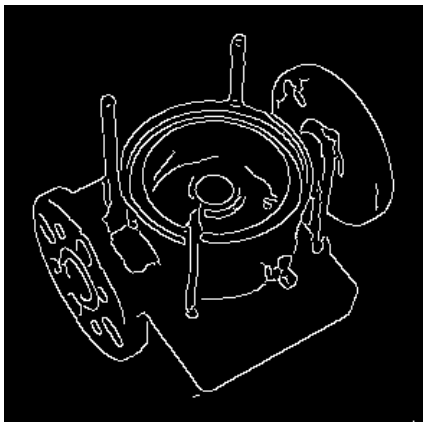


Abbildung 13: Ergebnis des Canny-Kantenerkennungs-Verfahrens

4.2 Fast-Fourier-Transformation

Die Fast-Fourier-Transformation ist ein Algorithmus zur schnellen Berechnung der diskreten Fourier-Transformation. Im Prinzip wird die Anzahl der, zur Berechnung einer DFT der Größe 2^n, nötigen $(2^n)^2$ komplexen Additionen und Multiplikationen auf $n * 2^n$ reduziert. (7)

Die diskrete Fourier Transformation (DFT) ordnet einem komplexen Vektor einen anderen komplexen Vektor der gleichen Länge zu, mathematisch: $f: \mathbb{C}^N \to \mathbb{C}^N, (a_1, \dots, a_n) \mapsto (b_1, \dots, b_n)$. Dabei ist die k-te Komponente des zugeordneten Vektors gegeben durch $b_k = \sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} \cdot a_j \, , k = 0, \dots, N-1$.

Es wird nun versucht zu erklären, warum die FFT effizienter ist als die normale DFT. Als Beispiel dient ein zu transformierender Vektor mit $4 = 2^2$ Elementen. Der vorige Zusammenhang kann dann auch als Matrix mal Vektor geschrieben werden (*Gl. 11*).

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \text{ mit } \omega = e^{-2\pi i \frac{1}{N}} \qquad \textbf{Gl. 11}$$

Da die Berechnung in einer Implementierung auch beginnen muss, kann man auch sagen es sind $(2^n)^2$ komplexen Additionen nötig, obwohl hier nur $2^n * (2^n - 1)$ ersichtlich sind (*Gl. 11*). Gruppiert man die geraden, sowie die ungeraden Vektorelemente in *Gl. 11* und beachtet man die Periodizität der komplexen Exponentialfunktion erhält man folgende Matrix (*Gl. 12*).

$$\begin{pmatrix} b_0 \\ b_2 \\ b_1 \\ b_3 \end{pmatrix} = \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & 1 & \omega^2 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^3 & \omega^2 & \omega \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}, \omega = e^{-2\pi i \frac{1}{N}} \qquad \textbf{Gl. 12}$$

Die Matrix in *Gl. 12* kann in zwei Matrizen aufgeteilt, also faktorisiert werden (*Gl. 13*).

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & 1 & \omega^2 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^3 & \omega^2 & \omega \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & \omega^2 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & \omega^2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & \omega^2 & 0 \\ 0 & \omega & 0 & \omega^3 \end{pmatrix} \qquad \textbf{Gl. 13}$$

Mit der ersten der beiden Matrizen in *Gl. 13* kann nun zuerst ein Zwischenergebnis berechnet werden, aus welchem sich anschließend mit der zweiten Matrix das Endergebnis ergibt (*Gl. 14*).

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & \omega^2 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & \omega^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0 + a_1 \\ a_1 + a_3 \\ a_0 - a_2 \\ (a_1 + a_3)\omega \end{pmatrix} \qquad \textbf{Gl. 14}$$

$$\begin{pmatrix} b_0 \\ b_2 \\ b_1 \\ b_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & \omega^2 & 0 \\ 0 & \omega & 0 & \omega^3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} c_0 + c_1 \\ c_0 - c_1 \\ c_2 + c_3 \\ c_2 - c_3 \end{pmatrix}$$

In *Gl. 14* kann die Reduktion der Anzahl der komplexen Additionen und Multiplikationen auf $2 * 2^2 = 8$ nachgezählt werden. Es scheint als wären es weniger Multiplikationen. In einer Implementierung des Verfahrens für längere Vektoren ist die Anzahl aber auch $n * 2^n$.

Der FFT-Algorithmus kann iterativ oder rekursiv implementiert werden. Um die Transformation in der richtigen Ordnung zu erhalten, ist eine Spiegelung der binären Darstellung nötig. Matlab beinhaltet eine mehrdimensionale FFT. Der Algorithmus wird für diese Arbeit nicht extra implementiert.

Midpoint-Kreis (Bresenham-Kreis)

Der Midpoint-Algorithmus wurde von Jack Bresenham 1964 bei IBM entwickelt und wird deshalb oft auch Bresenham-Algorithmus genannt. Das Midpoint-Verfahren ist ein Standardverfahren der Computergrafik um Rasterlinien und -kreise zu zeichnen.

Im Falle einer dicht liegenden Menge an Punkten, sind alle diejenigen Punkte auf der Kreislinie, welche folgende Bedingung erfüllen (*Gl. 15*).

$r = \sqrt{(x - x_m)^2 + (y - y_m)^2}$	**Gl. 15**

Wobei (x, y) die Koordinaten des Punktes und (x_m, y_m) die Koordinaten des Kreismittelpunktes sind.

Für eine Menge von diskreten Pixeln könnten alle Punkte, die folgende Bedingung (*Gl. 16*) erfüllen, als Kreispunkte definiert werden.

$r = \left\lfloor \sqrt{(x - x_m)^2 + (y - y_m)^2} + \frac{1}{2} \right\rfloor$	**Gl. 16**

Das konkrete Markieren der Kreispunkte mittels obiger Definition kann aber nur mit großem Rechenaufwand, gemeint sind viele Speicherzugriffe und Wertvergleiche, realisiert werden. Um einen Kreis in eine Bildmatrix zu zeichnen müssten alle Zeilen- und Spaltenadressen auf Übereinstimmung mit r geprüft werden. Falls *Gl. 16* erfüllt wird, dann muss der jeweilige Pixelwert geschrieben werden. In jedem Vergleichsschritt ist eine iterative Wurzelberechnung nötig, was Zeit kostet und deren Ergebnis einen größeren Speicherbedarf als die ganzzahlige Eingabe hat

Eine Implementierung, wie oben beschrieben, ist ein einfacher Lösungsansatz und könnte bei genügend verfügbarer Rechenleistung auch so implementiert werden. Besser ist es in jedem Fall den Midpoint-Algorithmus von Jack Bresenham einzusetzen.

Der Midpoint-Algorithmus zum Erzeugen eines Pixelkreises ist ein iteratives Verfahren, der in seiner besten Variante nur ganze Zahlen verarbeitet. Des Weiteren wird die hohe Symmetrie des Kreises ausgenützt.

Für die Beschreibung des Verfahrens wird das Koordinatensystem in den Kreismittelpunkt gelegt (*Abbildung 14*). Der Ursprung ist somit auch der Mittelpunkt der verwendeten $2 * r + 1 \times 2 * r + 1$ – Kreismatrix.

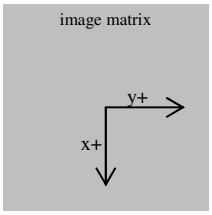


Abbildung 14: Koordinatensystem im Mittelpunkt der Kreismatrix

Die allgemeine Kreisgleichung mit Mittelpunkt $(0,0)$ ist *Gl. 17*.

$r^2 = x^2 + y^2$	**Gl. 17**

Diese Gleichung kann in eine explizite Form $f(x, y)$ (*Gl. 18*) umgeschrieben werden. Der Funktionswert der skalaren Funktion $f(x, y)$ gibt Auskunft darüber, ob ein Punkt mit dem Koordinatenvektor (x, y) auf dem Kreis liegt oder nicht.

$f(x, y) = x^2 + y^2 - r^2$	**Gl. 18**

Ein Punkt (x, y) liegt genau dann auf dem Kreis, falls $f(x, y) = 0$ ist, er ist innerhalb der Kreisfläche, wenn $f(x, y) < 0$ und außerhalb falls $f(x, y) > 0$.

Angenommen der Mittelpunkt ist genau ein Pixel und der Radius sollte acht Pixel betragen. In diesem Fall startet der Algorithmus an einem Punkt acht Punkte links vom Mittelpunkt, in der Horizontalen (*Abbildung 15*).

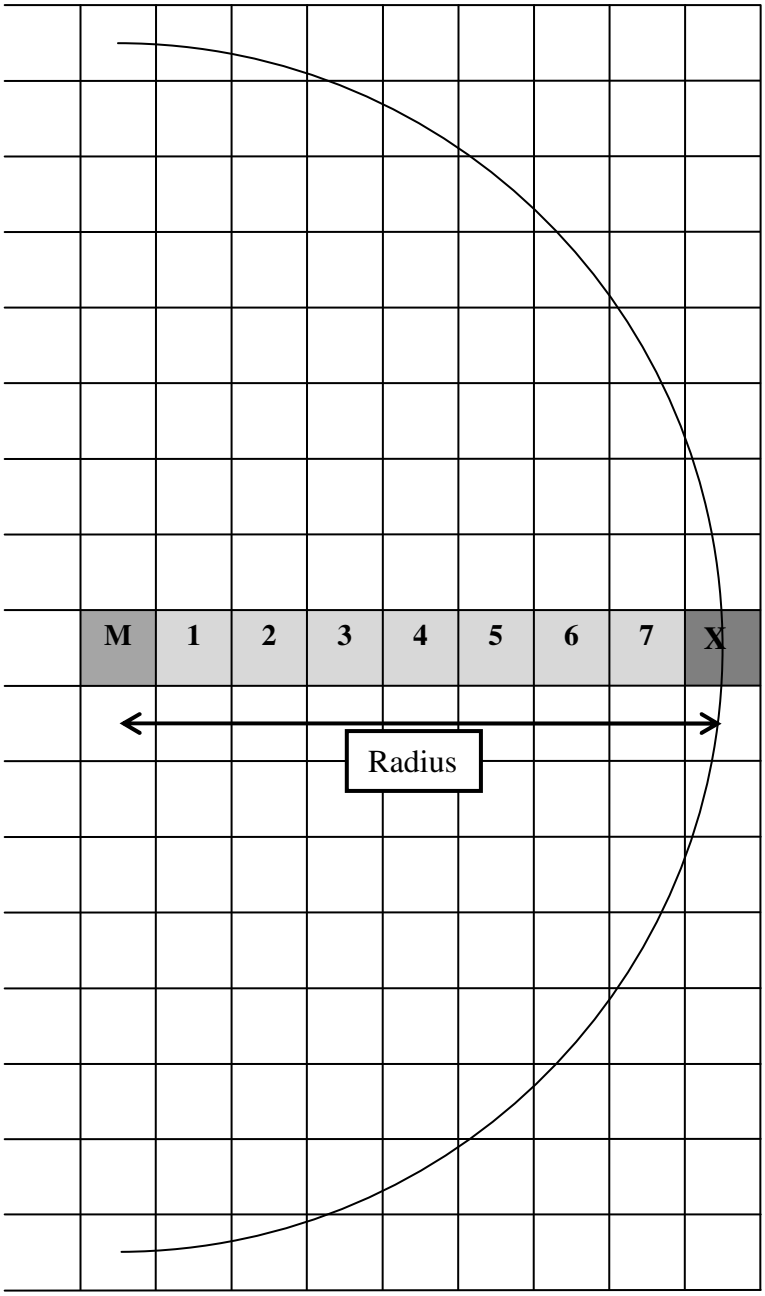


Abbildung 15: Startpunkt des Bresenham-Pixelkreis-Algorithmus (X)

In jeder Stufe des Algorithmus muss zwischen zwei Pixeln entschieden werden. Die Entscheidung findet auf Grund der Lage des Punktes zwischen den zu wählenden Pixeln, dem sogenannten midpoint satt.

Der Startpunkt des Algorithmus ist der Punkt $p_1 = (x_1 = 0, y_1 = 8)$. Der Radius ist also acht Pixel weit (*Abbildung 15*). Der Startwert des Midpoints wäre $m_0 = \left(0, \frac{15}{2}\right)$. Im ersten Schritt des Verfahrens liegt der Midpoint klarerweise innerhalb des Kreises und deshalb wird auch der Punkt bei $(x = 0, y = 8)$ initialisiert (*Abbildung 16*).

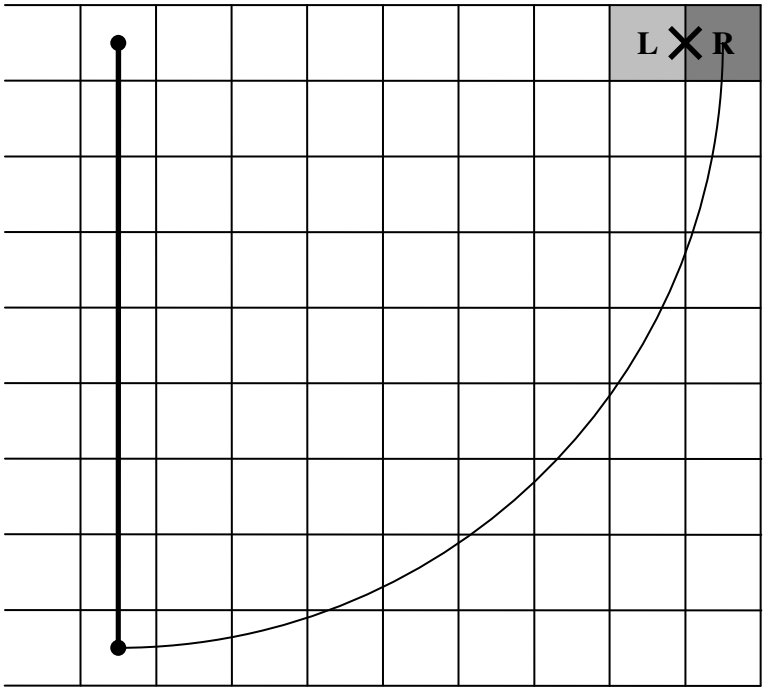


Abbildung 16: Das X markiert den Startwert des Midpoints

Ist der Midpoint innerhalb des Kreises soll der Pixel mit dem Label „R“ gewählt werden. Liegt der Midpoint hingegen außerhalb des Kreises ist der Pixel mit Aufschrift „L“ zu setzen. Zum Start des Verfahrens ist die Vorauswahl korrekt getroffen.

An diesem Punkt wird die Entscheidungsvariable d_i eingeführt. Die Entscheidungsvariable ist der Funktionswert der expliziten Kreisgleichung für die Koordinaten des Midpoints. Somit ergibt sich d_0 in konsistenter Weise (*Gl. 19*). Die im $i-1$-ten Schritt berechnete Entscheidungsvariable wird im i-ten Schritt zur Pixelwahl verwendet.

$d_0 = f\left(\left(0, \frac{15}{2}\right)\right)$	**Gl. 19**

Theoretisch wird der Wert des Midpoints zum nullten Schritt $m_0 = \left(0, \frac{15}{2}\right)$ von einem Punkt $p_0 = (x_0 = -1, y_0 = 8)$ im 0-ten Schritt abgeleitet, somit gibt es auch eine andere Möglichkeit d_0 auszudrücken bzw. zu berechnen (*Gl. 20*). Es sei erwähnt, dass d_0 innerhalb des 1-ten Schritts zur Entscheidungsfindung verwendet wird.

$d_0 = f\left(\left(x_0 + 1, y_0 - \frac{1}{2}\right)\right)$	**Gl. 20**

Es gilt $x_0 = -1$ ist und $y_0 = r$. d_0 reduziert sich unter Verwendung dieser Wert zu einem einfachen Ausdruck (*Gl. 21*).

$d_0 = f\left(\left(x_0 + 1, y_0 - \frac{1}{2}\right)\right) = f\left(\left(0, r - \frac{1}{2}\right)\right)$ $d_0 = 0^2 + \left(r - \frac{1}{2}\right)^2 - r^2 = r^2 - r + \frac{1}{4} - r^2$ $d_0 = \frac{1}{4} - r$	**Gl. 21**

Der Startwert ist für sinnvolle Radiuswerte mit Sicherheit negativ. $d_0 < 0$ hat die Wahl des rechten Punktes im ersten Iterationsschritt zu Folge, was wiederum mit der bereits getroffenen Vorauswahl konsistent ist. Der Algorithmus durchläuft eine Abfolge gleichartiger Schritte. d_1 ist der Startwert der Entscheidungsvariable, die im ersten Schritt, dem Initialisierungs-Schritt berechnet wird, und im zweiten Schritt zur Anwendung kommt (*Gl. 22*).

$d_1 = f\left(\left(x_1 + 1, y_1 - \frac{1}{2}\right)\right) = f\left(\left(1, r - \frac{1}{2}\right)\right)$ $d_1 = 1^2 + \left(r - \frac{1}{2}\right)^2 - r^2 = 1^2 + r^2 - r + \frac{1}{4} - r^2$ $d_1 = \frac{5}{4} - r$	**Gl. 22**

Um den Algorithmus zu starten ist die Berechnung von d_0 nicht nötig. Die Berechnung von d_0 macht gerade die erste Punktauswahl, bei Programmstart, plausibel.

Die Kern-Idee des Algorithmus liegt in einer iterativen Berechnung der Entscheidungsvariablen d. Wenn die Entscheidungsvariable (*decision variable*) negativ ist, liegt der Midpoint innerhalb des Kreises und der rechte Pixel wird als nächster Kreispunkt verwendet. Ist andererseits $d \geq 0$ liegt der Midpoint außerhalb des Kreises und der Algorithmus fährt mit der Wahl des linken Pixel fort (*Abbildung 16*).

$d_{i-1} < 0 \Rightarrow$ der Pixel mit der Aufschrift „R“ wird ausgewählt $d_{i-1} \geq 0 \Rightarrow$ der Pixel mit der Aufschrift „L“ wird ausgewählt	**Gl. 23**

Befindet sich der Algorithmus in der i-ten Stufe ist die Entscheidungsvariable d_{i-1} bekannt. Innerhalb dieser Stufe wird je nach Vorzeichen von d_{i-1} entweder der „R"-Pixel oder der „L"-Pixel gewählt. Wie lässt sich die Entscheidungsvariable d_i für den nächsten, also den $i+1$-ten Schritt berechnen? Klarerweise nimmt die x-Koordinate des Midpoints in jedem Schritt immer um eins zu. Die y-Koordinate des Midpoints nimmt hingegen nur bei Auswahl des „L"-Punktes im momentanen Schritt um eins ab.

Die Differenz zwischen d_i und d_{i-1} wird mit Δd bezeichnet (*Gl. 24*)

$\Delta d = d_i - d_{i-1} \Leftrightarrow d_i = d_{i-1} + \Delta d$	**Gl. 24**

Angenommen der „R"-Punkt wird aufgrund des Zutreffens von $d_{i-1} < 0$ im i-ten Schritt gewählt, dann sind seine Koordinaten einfach zu berechnen (*Gl. 25*).

$x_i = x_{i-1} + 1$ $y_i = y_{i-1}$	**Gl. 25**

Um sicherzustellen, dass bei Verwendung des selben Speicherortes für alle x_k bzw. y_k, immer noch x_{i-1} und y_{i-1} verwendet werden, findet das Erneuerern der Entscheidungsvariable von $d_{i-1} \longrightarrow d_i$ in einer Implementierung des Algorithmus vor dem Inkrementieren der Koordinaten statt. Für die Aktualisierung von d_{i-1} auf d_i muss Δd berechnet werden (*Gl. 26*).

$\Delta d = d_i - d_{i-1}$ $\Delta d = f\left(x_i + 1, y_i - \frac{1}{2}\right) - f\left(x_{i-1} + 1, y_{i-1} - \frac{1}{2}\right)$ $\Delta d = f\left(x_{i-1} + 2, y_{i-1} - \frac{1}{2}\right) - f\left(x_{i-1} + 1, y_{i-1} - \frac{1}{2}\right)$ $\Delta d = (x_{i-1} + 2)^2 + \left(y_{i-1} - \frac{1}{2}\right)^2 - r^2 - (x_{i-1} + 1)^2 - \left(y_{i-1} - \frac{1}{2}\right)^2 + r^2$ $\Delta d = (x_{i-1} + 2)^2 - (x_{i-1} + 1)^2$ $\Delta d = {x_{i-1}}^2 + 4x_{i-1} + 4 - {x_{i-1}}^2 - 2x_{i-1} - 1$ $\Delta d = 2x_{i-1} + 3$ $d_i = d_{i-1} + 2x_{i-1} + 3$	**Gl. 26**

Wird im i-ten Schritt der Punkt mit der Beschriftung „R“ gewählt so ist die neue Entscheidungsvariable für den $i+1$-ten Schritt gegeben durch $d_i = d_{i-1} + 2x_{i-1} + 3$.

Einsetzen von d_0 und d_1 prüft *Gl. 26* auf Konsistenz (*Gl. 27*).

$d_1 = d_0 + 2x_0 + 3$ $\frac{5}{4} = \frac{1}{4} - r + 2(-1) + 3 = \frac{5}{4} - r$	**Gl. 27**

Wird aufgrund des Zutreffens von $d_{i-1} > 0$ der „L“-Punkt im i-ten Schritt gewählt, muss die Entscheidungsvariable etwas anders berechnet werden. Die neuen Punktkoordinaten sind dann (*Gl. 28*):

$x_i = x_{i-1} + 1$ $y_i = y_{i-1} - 1$	**Gl. 28**

Auch die y-Koordinate wird verändert, sie wird um eins erniedrigt und d_i berechnet sich dann wie in *Gl. 29*.

$\Delta d = d_i - d_{i-1}$ $\Delta d = f\left(x_i + 1, y_i - \frac{1}{2}\right) - f\left(x_{i-1} + 1, y_{i-1} - \frac{1}{2}\right)$ $\Delta d = f\left(x_{i-1} + 2, y_{i-1} - \frac{3}{2}\right) - f\left(x_{i-1} + 1, y_{i-1} - \frac{1}{2}\right)$ $\Delta d = (x_{i-1} + 2)^2 + \left(y_{i-1} - \frac{3}{2}\right)^2 - r^2 - (x_{i-1} + 1)^2 - \left(y_{i-1} - \frac{1}{2}\right)^2 + r^2$ $\Delta d = {x_{i-1}}^2 + 4x_{i-1} + 4 + {y_{i-1}}^2 - 3y_{i-1} + \frac{9}{4} - {x_{i-1}}^2 - 2x_{i-1} - 1 - {y_{i-1}}^2 + y_{i-1} - \frac{1}{4}$ $\Delta d = 2x_{i-1} - 2y_{i-1} + 5$ $d_i = d_{i-1} + 2x_{i-1} - 2y_{i-1} + 5$	**Gl. 29**

Beide Fälle werden grafisch dargestellt. Der Fall $d_{i-1} < 0$ tritt im dritten Iterationsschritt auf. Welchen Wert hat $d_{i-1} = d_{3-1} = d_2$? Die Berechnung kann mit d_1 iterativ erfolgen (*Gl. 30*).

$d_2 = d_1 + 2x_1 + 3 = \frac{5}{4} - r + 2 * 0 + 3 = \frac{17}{4} - r$ $d_2 = \frac{17}{4} - 8 = -\frac{15}{4} < 0$	**Gl. 30**

Also wird in Iterationsschritt drei der Pixel mit der Aufschrift „R“ gewählt (*Abbildung 17*).

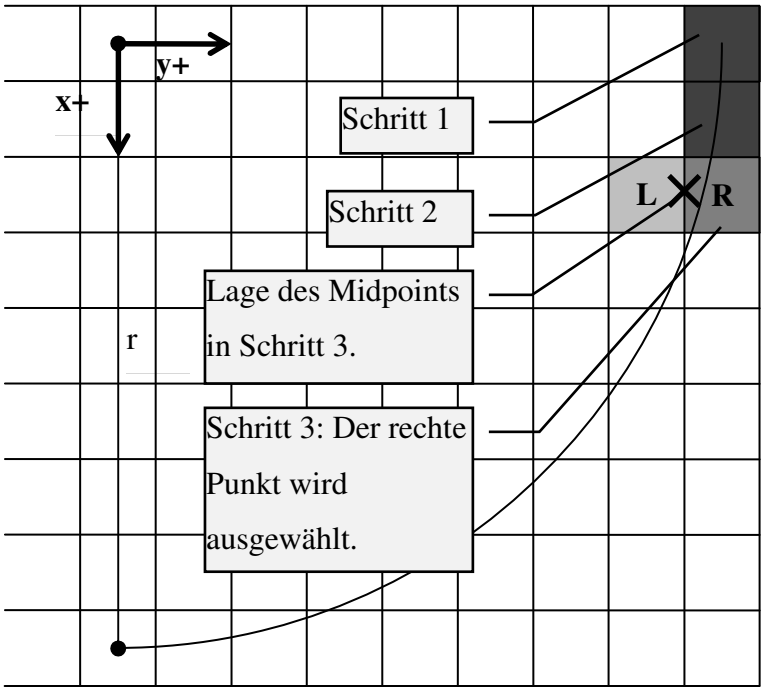


Abbildung 17: Iterationsstufe 3. Das Kreuz markiert den Midpoint. Der „R“-Pixel wird gesetzt.

Was passiert nun im nächsten Schritt vier? Innerhalb von Schritt drei wird noch die Entscheidungsvariable d_3 berechnet (*Gl. 31*).

$d_3 = d_2 + 2x_2 + 3 = -\frac{7}{4} + 2 * 1 + 3 = \frac{13}{4}$ $d_3 = \frac{13}{4} > 0$	**Gl. 31**

Mit $d_3 > 0$ wird im vierten Schritt der linke Punkt gewählt (*Abbildung 18*).

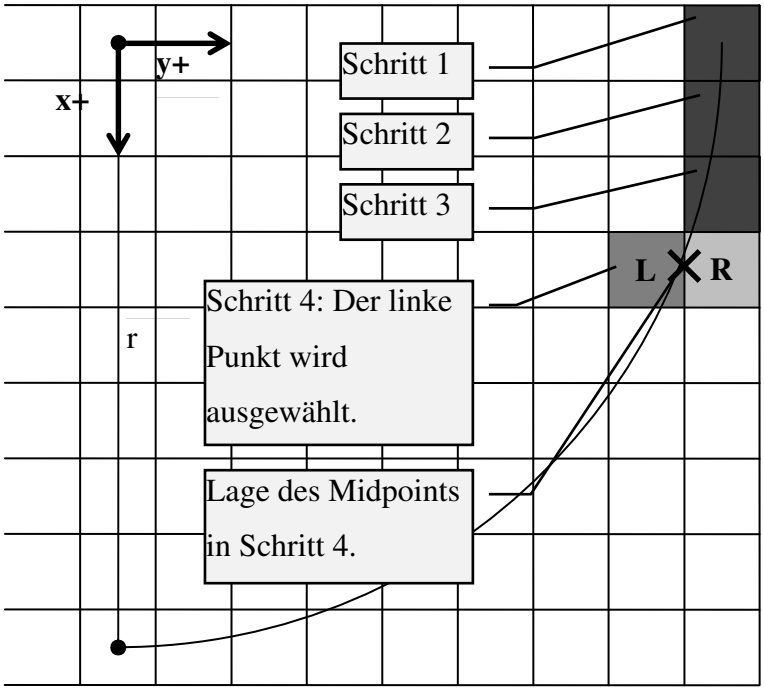


Abbildung 18: Iterationsschritt 3

Im vierten Schritt liegt der Midpoint außerhalb des Kreises. Die Wahl des linken Punktes ist also logisch. Der vierte Schritt berechnet die Variable d_4 nun mit *Gl. 32*.

$d_i = d_{i-1} + 2x_{i-1} - 2y_{i-1} + 5$ $d_4 = \frac{13}{4} + 2 * 2 - 2 * 8 + 5 = \frac{13}{4} - 7 = -\frac{15}{4}$ $d_4 = -\frac{15}{4} < 0$	**Gl. 32**

Folgender Iterationsschritt fünf wird also wiederum den rechten Punkt wählen. Der Midpoint ändert in Schritt vier zum ersten Mal seit Beginn des Verfahrens seine y-Koordinate.

Aufgrund der hohen Symmetrie eines Kreises ist es ausreichend den Kreisbogen nur bis zu einem Winkel von $45°$ zu berechnen. Der Algorithmus bricht also dann ab wenn die Bedingung $y > x$ nicht mehr erfüllt ist.

Um nur mit ganzen Zahlen rechnen zu müssen kann d_1 als $1 - r$ initialisiert werden.

5 Eigene Programm Implementierungen

Es werden fünf verschiedene Programme implementiert.

- Kreismittelpunktsiteration mit vorheriger statistischer Bereichseinschränkung
- Kanteniteration mit verschiedenen Bildgrößen
- Template Matching Programme
    - Echte Template Matching Methode
    - Feature basierte Template Matching Methode
- Daugman-Integro-Differential-Operator basiertes Programm

Genauer untersucht und beschrieben werden nur die Template-Matching Programme und das Daugman-Integro-Differential-Operator basierte Programm. Innerhalb der Template-Matching Verfahren ist das feature-Matching letztendlich als erfolgreichere Lösung hervorgegangen, dennoch wird auf die echte Template-Methode eingegangen um die Zusammenhänge zu verdeutlichen.

Beide Programme werden unter Verwendung der Bereichseinschränkung (*region of interest*) mit einander verglichen. Ohne die Einschränkung sind vom Rechenaufwand bedingt nur Bildhöhen von 640 Pixel möglich. Wobei die Bildbreite je nach Seitenverhältnis gegeben ist. Die Samples werden auf eine Bildhöhe von 64 bzw. 128 Pixel reduziert um die *region of interest* zu finden. Solange die *region of interest* die Größe 640x640 nicht überschreitet, kann die Berechnung mit aktivierter Bereichseinschränkung Bilder der Höhe 1024 Pixel bearbeiten. Die Vorverarbeitung und die Bereichseinschränkung laufen in beiden Programmen ähnlich ab. Innerhalb der Vorverarbeitung werden Schliffspuren entfernt und ein ungünstiger Helligkeitsverlauf ausgeglichen.

5.1 Vorverarbeitung

In beiden Programmen läuft vor der Bereichseinschränkung dieselbe Vorverarbeitung des Bildes ab. Einerseits, wird der das Bild überlagernde Helligkeitsverlauf so gut wie möglich ausgeglichen und andererseits werden mittels Fourier-Transformation Schleifspuren entfernt.

5.1.1 Helligkeits-Anpassung

Die Ermittlung der Helligkeit kann entweder durch morphologisches Öffnen (Erosion gefolgt von Erweiterung) des Graustufenbildes, oder aber auch durch starke Gaussglättung des Graustufenbildes erfolgen und anschließend können die jeweiligen Hintergrundüberlagerung von den Bildpixeln abgezogen werden. Es wird zwischen hellen

und dunklen Bildern unterschieden und je nachdem das Bild durch den Helligkeitsausgleich entweder aufgehellt oder abgedunkelt.

5.1.1.1 morphologisches Öffnen

Um ein Graustufenbild morphologisch zu öffnen, muss zunächst ein Strukturelement definiert werden. Ein einfaches Strukturelement könnte folgende Gestalt haben (*Tabelle 4*).

0	**0**	**1**	**0**	**0**
0	**1**	**1**	**1**	**0**
1	**1**	**1**	**1**	**1**
0	**1**	**1**	**1**	**0**
0	**0**	**1**	**0**	**0**

Tabelle 4: Strukturelement

Wie schon erwähnt ist das morphologische Öffnen eine Erosion (*erosion*) gefolgt von einer Erweiterung (*dilation*) des Bildes. Die morphologische Öffnung eines Graustufenbildes funktioniert ähnlich wie die morphologische Öffnung eines binären Bildes. Im Eingabebild wird jeder Pixel und dessen Umgebung je nach Art des Strukturelements untersucht. Im Falle eines Strukturelements wie in *Tabelle 4* werden alle Nachbarpixel an denen Eins steht untersucht. Für die Erweiterung wird im Ausgabebild der Maximalwert der Umgebungspixel gesetzt (*Abbildung 19*).

116 **17** **28** **33** **45** **126** **157** **62** **21** **68** **132** **130** **64** **15** **57** **150** **141** **139** **60** **53** **142** **138** **134** **144** **14**	* * * * * * * * * * * * **141** * * * * * * * * * * * *
Eingabebild	Ausgabebild

Abbildung 19: morphologisches Erweitern

Die Erosion läuft ähnlich der morphologischen Erweiterung ab, der einzige Unterschied liegt darin, dass der kleinste Wert im Ausgabebild gesetzt wird.

Wenn das Bild durch einen Helligkeitsverlauf überlagert wird und einzelne begrenzte Bereiche im Bild andere Helligkeitswerte haben, dann werden diese Bereiche durch die Erosion unterdrückt, durch die anschließende Erweiterung bleibt nur noch der

Hintergrundhelligkeitsverlauf zurück. *Abbildung 20* hat eine nicht einheitliche Hintergrundbeleuchtung. Durch das morphologische Öffnen verschwinden die länglichen Elemente und die Hintergrundbeleuchtung kann abgeschätzt werden.

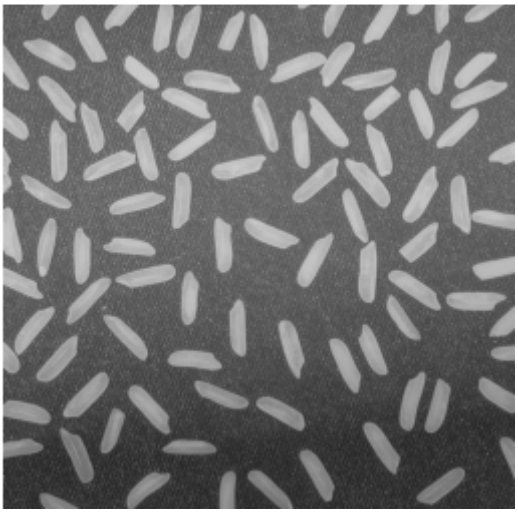


Abbildung 20: Hintergrund Helligkeitsverlauf

Der geschätzte Helligkeitsverlauf hat die folgende Gestalt (*Abbildung 21*). Wird dieser Verlauf vom Eingabebild abgezogen, verschwindet die uneinheitliche Hintergrundbeleuchtung.

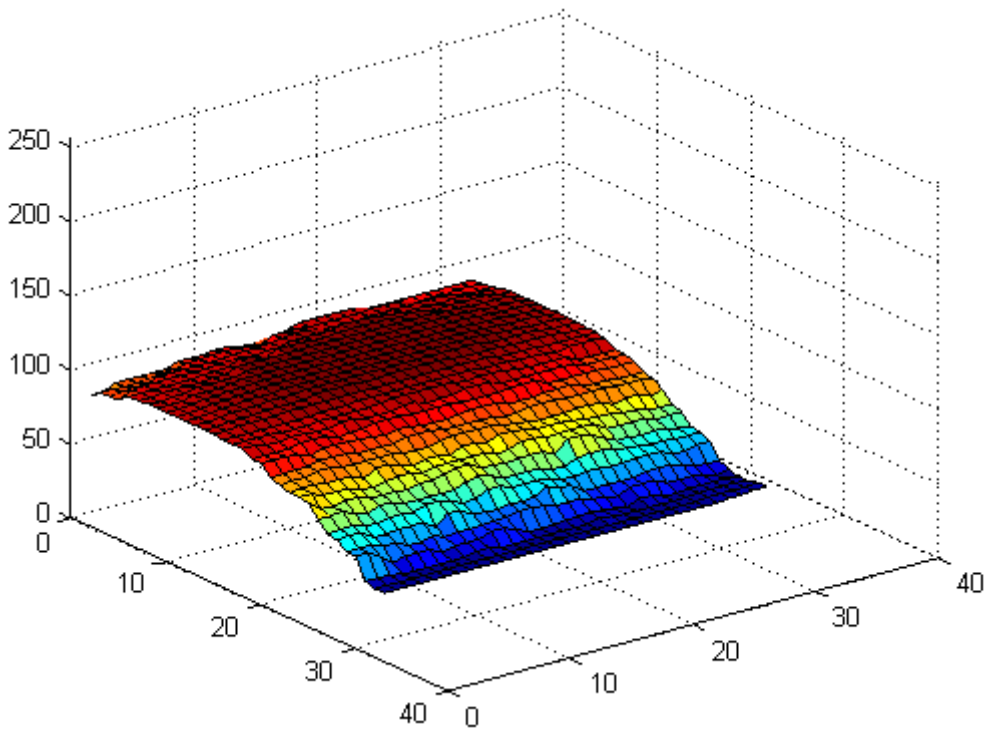


Abbildung 21: Hintergrund Helligkeitsverlauf

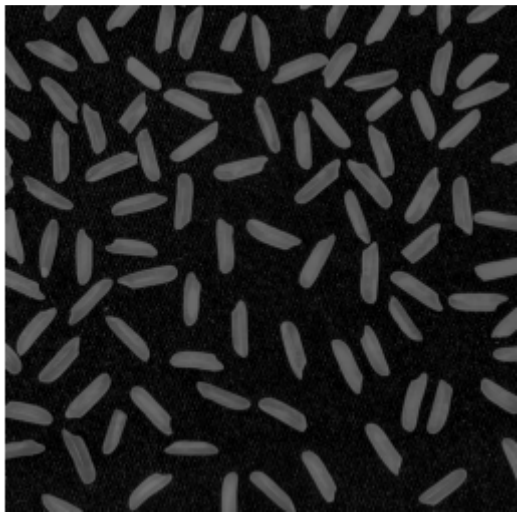


Abbildung 22: Bild ohne Hintergrundbeleuchtung

Dieses Verfahren kann natürlich nur beschränkt auf die Brinell-Bilder angewandt werden. Die Hoffnung besteht darin Oberflächenunebenheiten, die als Flecken zu sehen sind, bzw. den Abdruck weg zu bekommen und nur den Helligkeitsverlauf zu behalten. Auf ein Brinellabdruck-Bild angewandt erreicht man je nach Größe des Strukturelements verschiedene Ergebnisse. Für das nachfolgende Bild (*Abbildung 23*) wird ein diamantförmiges Strukturelement mit einem Durchmesser von 15 verwendet.

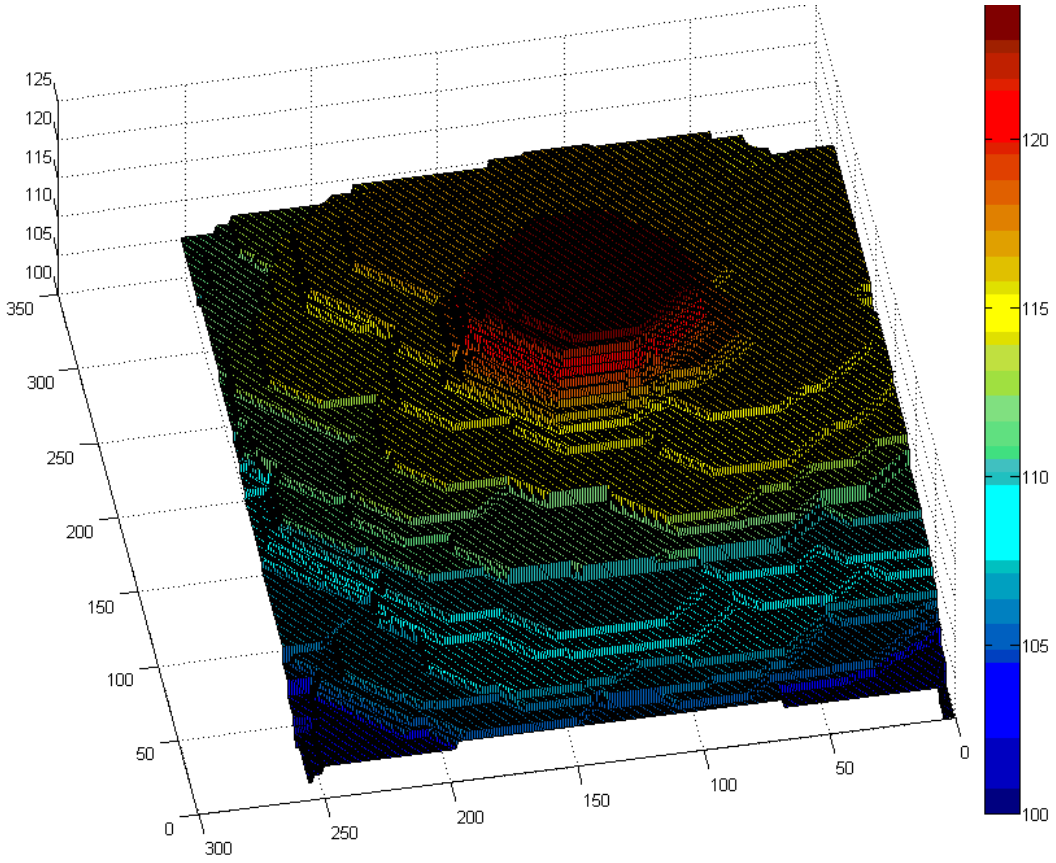


Abbildung 23: Ergebnis des morphologischen Öffnens eines Brinellbildes

In obiger *Abbildung 23* ist zu erkenne, dass sich der Abdruck nicht ganz entfernen lässt. Das Verfahren ist vielleicht weniger gut für diese Bilder geeignet. Andererseits wird aber ein Helligkeitsverlauf sichtbar.

5.1.1.2 Gaussglättung

Eine weitere Möglichkeit den Helligkeitsverlauf abzuschätzen ist durch eine starke Gaussfilterung des Bildes gegeben. Die Filterung wird im Bildbereich durchgeführt und der gauss‘sche Lowpass muss entsprechend dimensioniert sein. Lässt man die verschiedenen Brinellbilder also durch einen Gaussfilter mit genügender Größe laufen, ergibt sich ein Ergebnis wie in nachfolgender *Abbildung 24*. Dieser Verlauf spiegelt die Beleuchtung wieder.

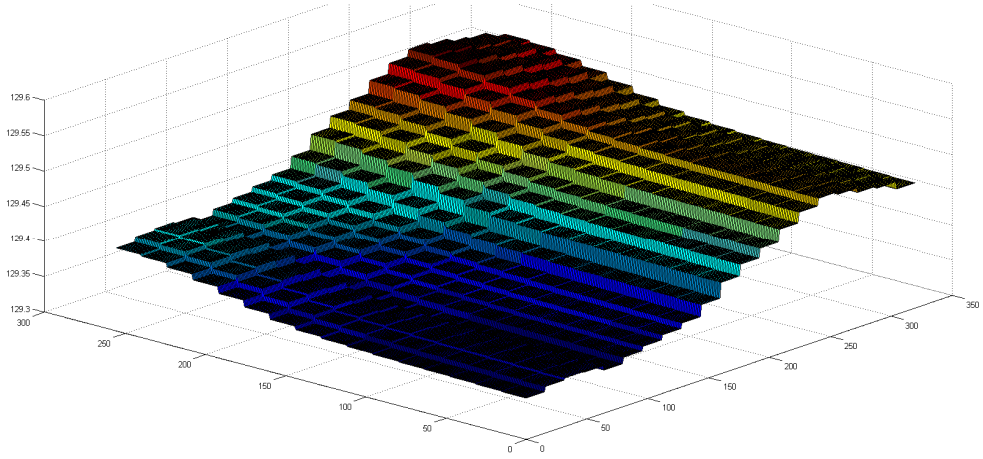


Abbildung 24: überlagerte Beleuchtung

Je nachdem ob der Mittelwert aller Bildpunkte größer oder kleiner als 128 ist wird das Bild als Hell oder Dunkel eingestuft. In einem hellen Bild werden die Verläufe in den *Abbildung 23* und *Abbildung 24* abgezogen in einem dunklen Bild dazu gezählt.

Grundsätzlich ist anzumerken, dass sich die Helligkeitsunterschiede durch die Beleuchtung im Bild innerhalb einer Wertespanne von 3-8 bewegen. – Das ist nicht sehr viel und somit ist der Effekt des Helligkeitsausgleichs auch nicht sehr gut zu bemerken.

5.1.2 Entfernen der Schleifspuren

Es kann durchaus passieren, dass Abdruckbilder mit sehr starken Schleifspuren verarbeitet werden müssen. Diese Spuren sind durch Oberflächenbearbeitung jeglicher Art hervorgerufen. Ein Beispiel für eine solche starke, den gesamten Abdruck überlagernde Bearbeitung ist in nachfolgender *Abbildung 25* gezeigt.

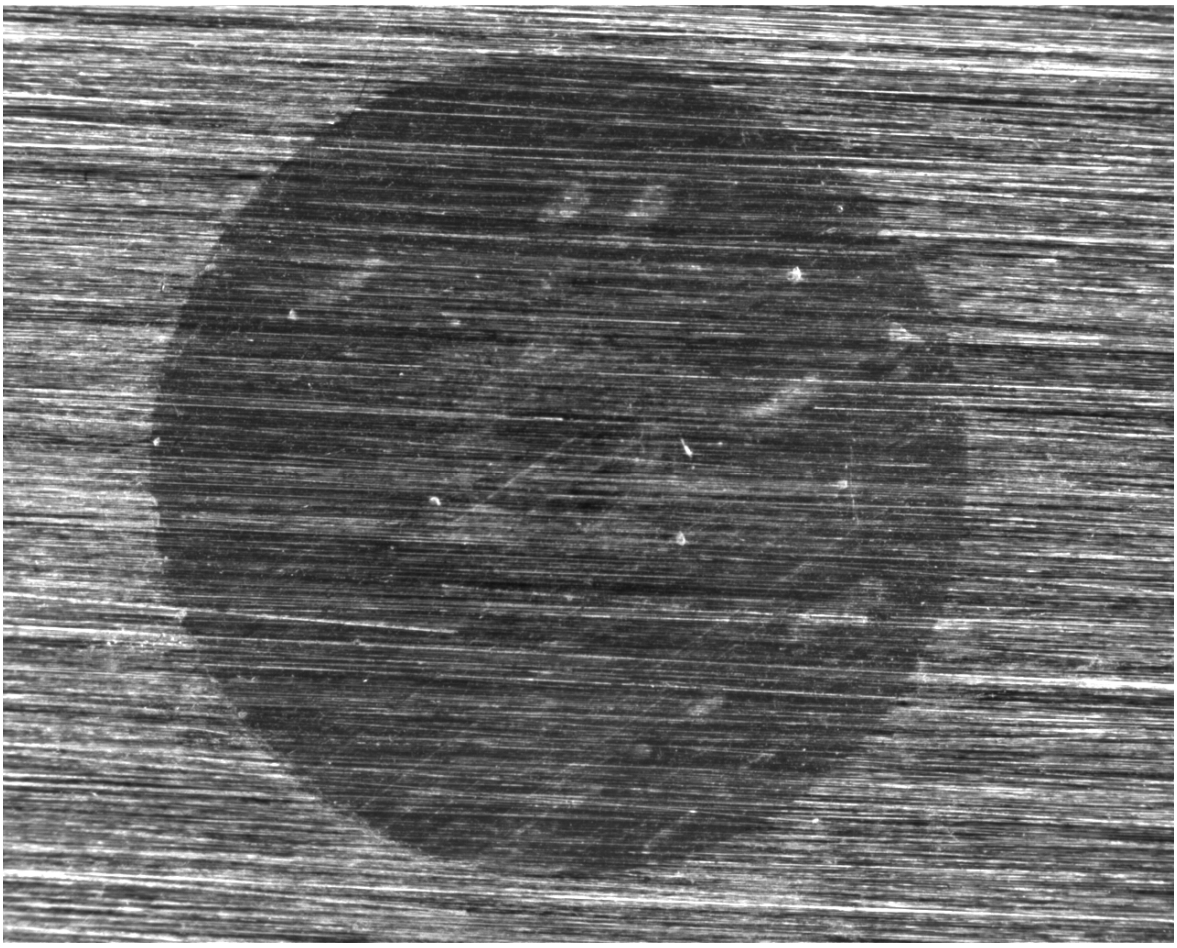


Abbildung 25: Abdruck mit stark Form-überlagernden Schleifspuren

Die Bearbeitungsspuren können durch Fouriertransformation des Bildes, anschließende Bearbeitung des Spektrums und darauf folgende Rücktransformation entfernt werden.

In Matlab ist eine diskrete Fouriertransformation standardmäßig enthalten. Die Implementierung verwendet die sogenannte Fast-Fourier-Transformation. Es sollte auf ein quadratisches Bild geachtet werden. Des Weiteren sollten die Seitenlängn nicht prim zu einander und wenn möglich Potenzen von 2 sein.

Im Fourierspektrum sind es genau die Frequenzanteile senkrecht zur Bearbeitungsrichtung, die entfernt werden müssen. Nachfolgend ist das Spektrum des Abdruckbildes in *Abbildung 25* dargestellt. Es sei noch angemerkt, dass in Matlab ein Shifting der Transformation nötig ist um die Nullfrequenzen in den Mittelpunkt des Spektrum-Koordinatensystems zu verschieben.

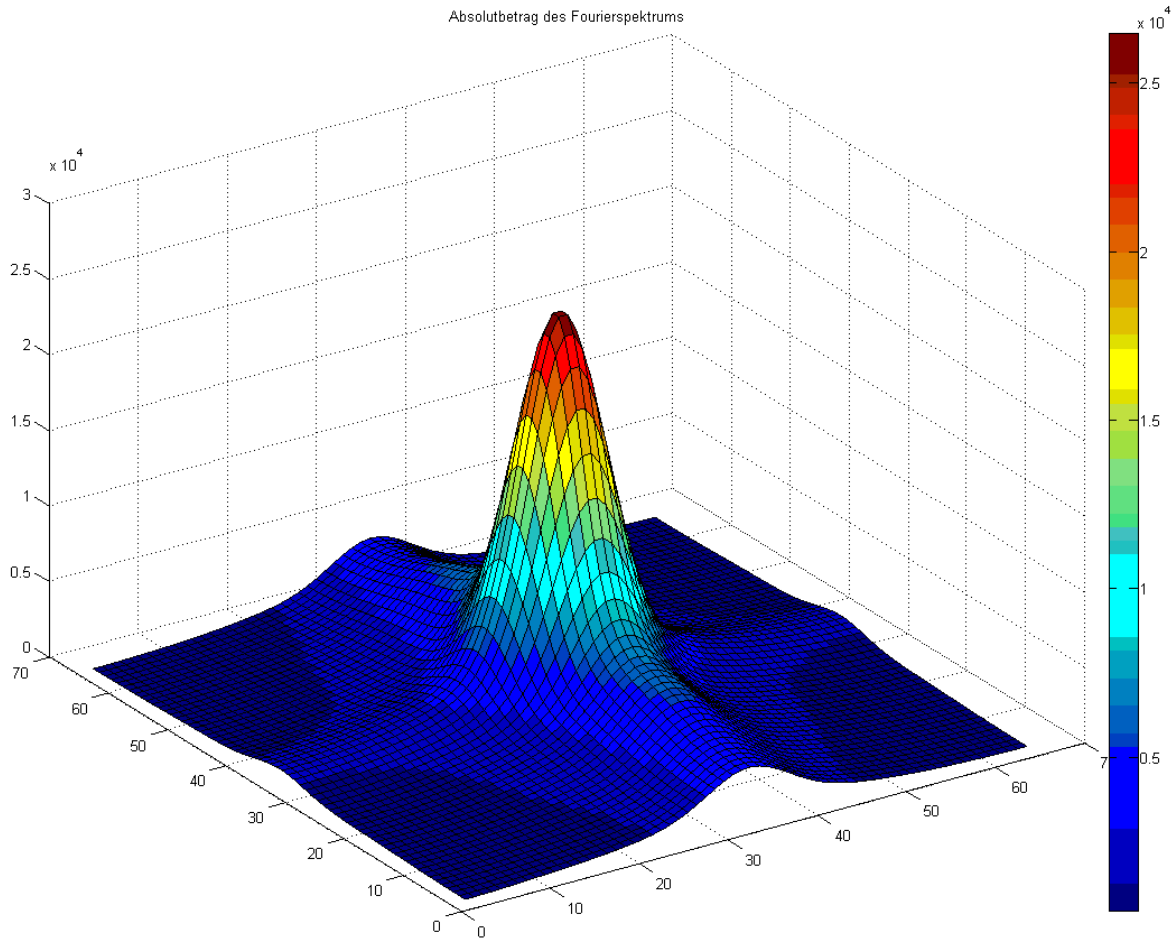


Abbildung 26: Fourierspektrum von Abbildung 26

Tatsächlich handelt es sich in *Abbildung 26* um ein stark geglättetes Spektrum. Würde das Spektrum nicht mit Low-Pass-Filtern geglättet werden, dann könnte man keine schöne Kontur erkennen. Die Frequenzen die nun durch die Oberflächenbearbeitung hervorgerufen werden bilden den auffälligen Kamm, der sich quer durch das Spektrum zieht. Die eher kleine Aufwölbung, die auch das ganze Spektrum von links nach rechts durchläuft kann vertikalen Bildkanten zugeschrieben werden.

Um die Frequenzen der Oberflächenbearbeitung zu entfernen muss ein entsprechender Low-Pass-Filter gestaltet werden. Die Frequenzen im Zentrum des Spektrums müssen erhalten bleiben, ansonsten folgt einen Helligkeitsverlust oder die Beschädigung des Kreises.

Der Low-Pass-Filter sollte die Frequenzen des Kamms möglichst auf null reduzieren, aber das andere Spektrum größtenteils unangetastet lassen. Um einen Filter dahingehend zu gestalten, muss die Ausrichtung des Kamms einigermaßen genau bekannt sein. Diese automatisch zu bestimmen ist nicht ganz einfach. Im Falle mancher Spektren könnte sie

durch Thresholding und anschließende lineare Hough-Transformation detektiert werden. Das ist aber nur bedingt möglich und führt bei manchen Spektren sicher zu Misserfolg. Eine andere Möglichkeit die Richtung der Bearbeitung herauszufinden ist die Betrachtung der Höhenlinien. Dazu sollte aber ein logarithmisch skaliertes Fourier-Spektrum herangezogen werden, so wird eine stärkere Gewichtung kleinerer Frequenzanteile und eine Abschwächung höherer sowie eine stärkere Exzentrizität der Höhenlinien erreicht. In dieser Implementierung wird ein bestimmter Höhenbereich durchlaufen und die Höhenlinie mit der maximalen linearen Exzentrizität verwendet. Nachfolgend wird das logarithmisch skalierte Spektrum und seine Höhenlinien dargestellt (*Abbildung 27, Abbildung 28*).

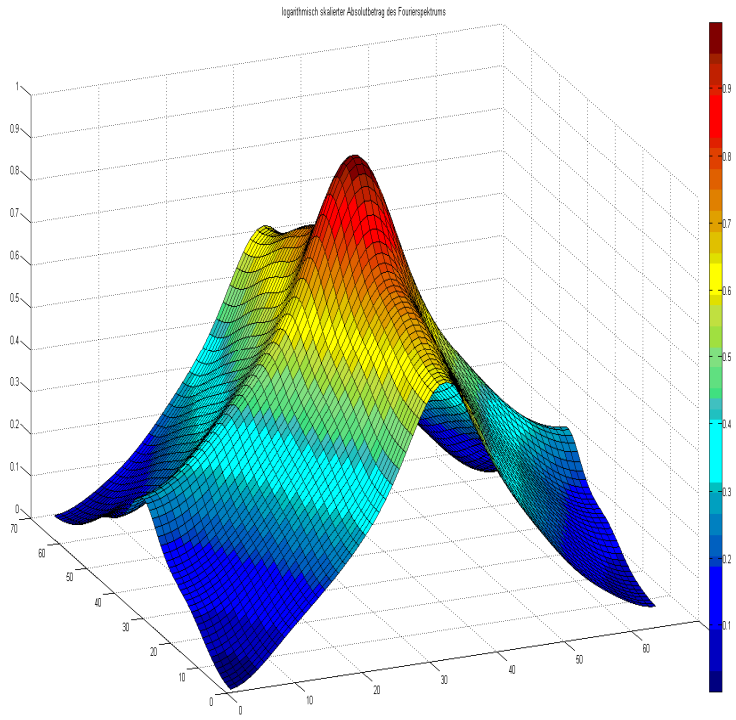


Abbildung 27: logarithmisch skalierter Absolutbetrag des Fourierspektrums

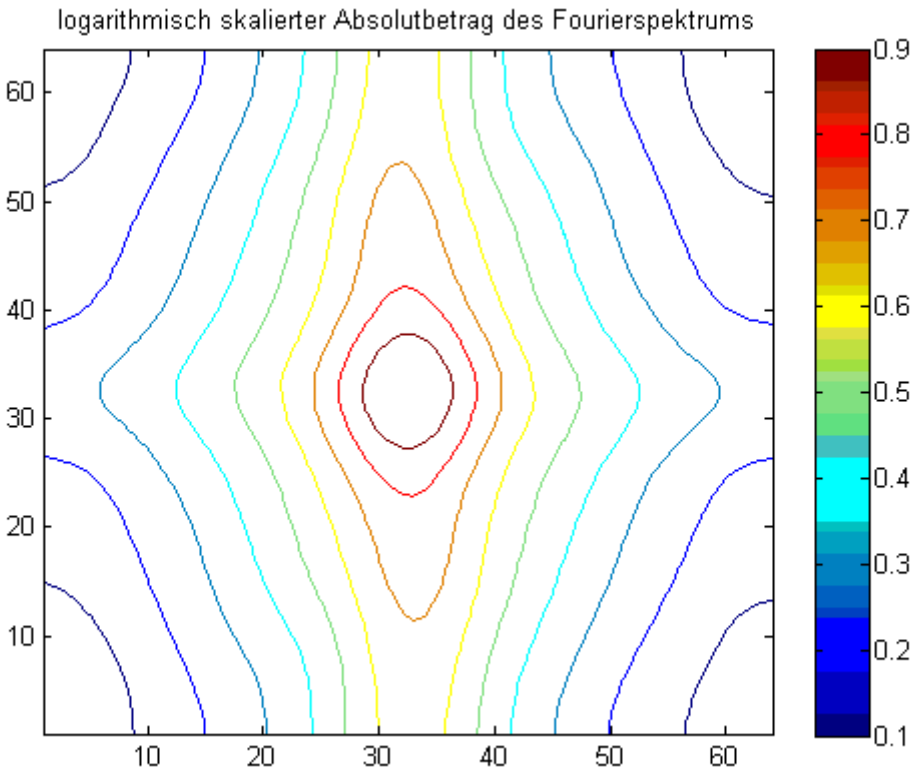


Abbildung 28: Höhenlinienverlauf des Spektrums

Wird eine Höhenlinie des Spektrums herausgenommen kann dieser Höhenlinie eine Ellipse umschrieben werden und auf diese Weiße die Ausrichtung der Hauptachse ermittelt werden. Mit Matlab kann die Orientierung einer geschlossen Höhenlinie mit *regionprops* ermittelt werden. In nachfolgender *Abbildung 29* wird die durch die Matlab Methode *regionprops* detektierte Orientierung dargestellt.

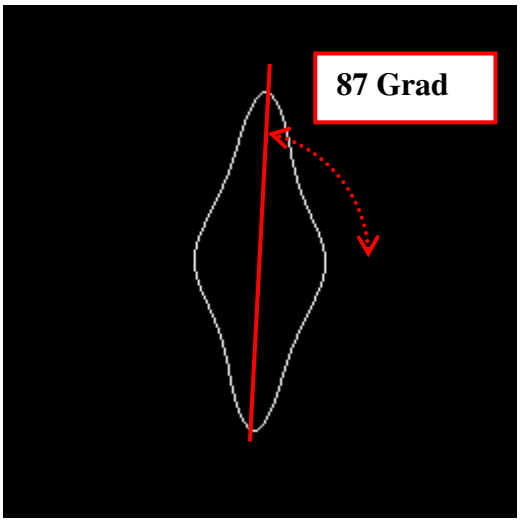


Abbildung 29: Ausrichtung der Höhenlinie

Mit der Ausrichtungsinformation kann ein nur auf diesen Bereich wirkender Low-Passfilter erstellt werden. Es handelt sich um einen Gaussfilter, der auf einen linearen Bereich eben genau diese Frequenzen der Oberflächenbearbeitung eingeschränkt ist. Die Gestalt des Filters ist in *Abbildung 30* zu sehen.

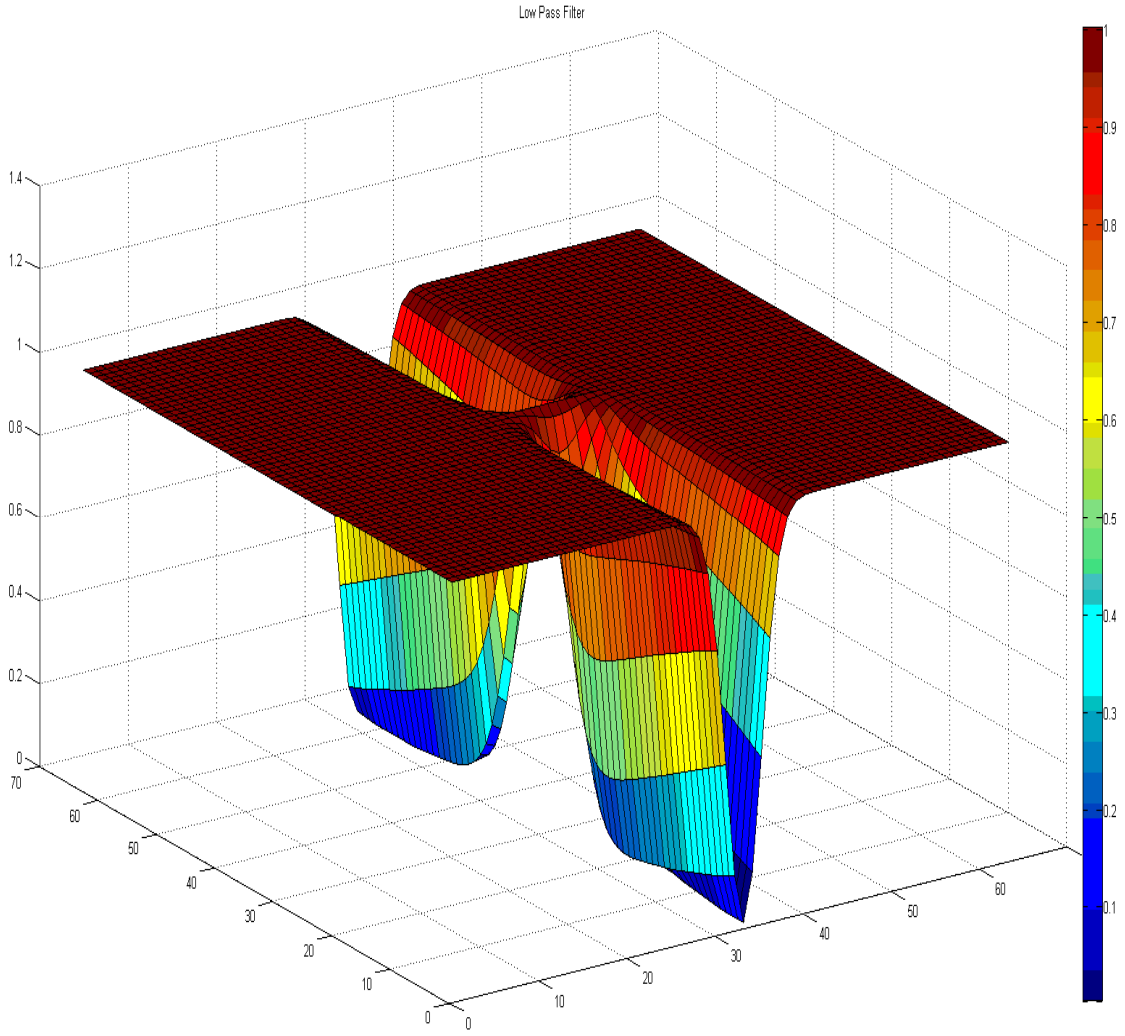


Abbildung 30: Low-Pass Filter

Der Lowpass entsteht durch die Bildung des äußeren Produktes zweier Gausskurven. Anschließend wird er noch in die entsprechende Richtung gedreht. Die Richtung wird aus der vorher gezeigten Höhenlinienausrichtung ermittelt. Nachfolgend wird nochmals die Konstruktion des Low-Pass Filters dargestellt (*Abbildung 31*).

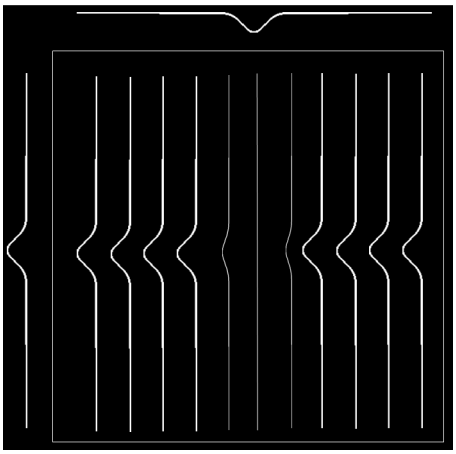
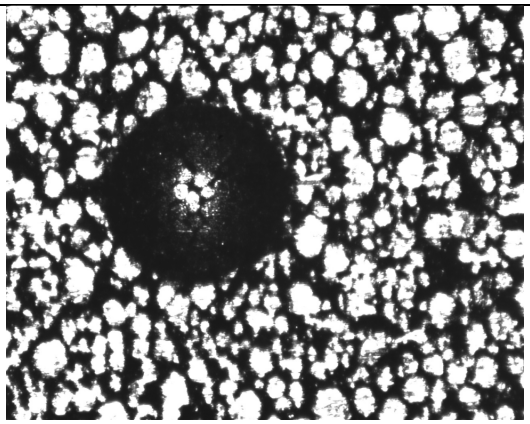
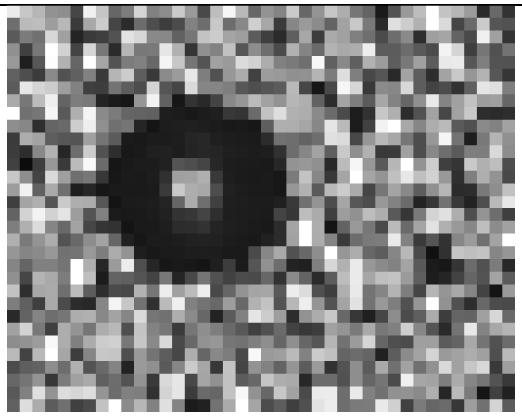


Abbildung 31: äussers Produkt zwischen Gausskurven

5.2 Region of interest

Beide Programme suchen durch das eigentlich verwendete Suchverfahren, sprich durch Template-Matching oder den Daugman-Integro-Differentialoperator den Bereich des Abdrucks.

Das Bild wird für die Bereichssuche verkleinert. Damit wird nicht nur der Rechenauf reduziert, sondern das Template-Matching Programm wird auch Immun gegen Texturen der folgenden Art (*Tabelle 5*).

Originalbild	interpoliertes verkleinertes Bild
	
Kantenbild	besseres Kantenbild

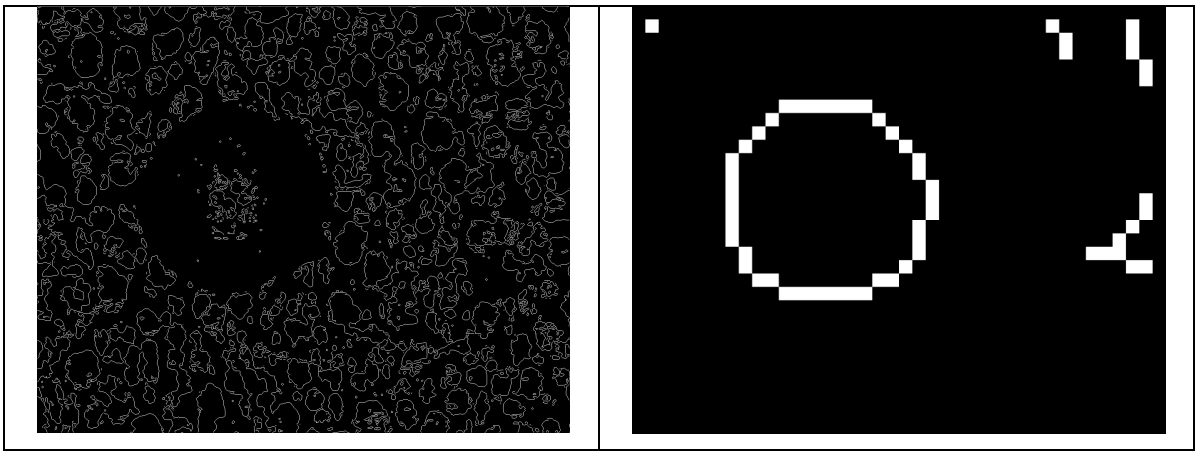


Tabelle 5

Innerhalb der region of interst-Suche wird ein Kreis gefunden, danach wird eine Maske für die region of interest erstellt und der entsprechende Bereich weiterverarbeitet. – Das bedeutet aus dem originalen Bild wird der Abdruckbereich ausgeschnitten, dazu ist eine Vergrößerung der binären Maske auf die Größe des Originalbildes nötig.

In dem Kantenbild des verkleinerten Bildes wird durch das Template-Matching-Verfahren ein Kreis gefunden. Auch der Daugman-Integro-Differential-Operator findet im verkleinerten Bild einen Kreis. Die umschreibenden Rechtecke dieser Kreise werden, um den Abdruckbereich nicht abzuschneiden, etwas erweitert und als region of interst verwendet.

5.3 Kernfunktionalität des Template-Matching Programms

Innerhalb des Template Matchings wurden zwei Verfahren realisiert. Das eine Verfahren basiert auf echten Templates und das andere auf der Feature-Matching Methode. Unter Feature-Matching wird im Allgemeinen das Matching von bereits herausgelesenen Eigenschaften verstanden. In diesem Falle wird das Kantenbild einem Template-Matching Verfahren unterzogen.

Beide Methoden sind Teil derselben Klasse. Die echte Template-Matching-Methode wird aber momentan nicht verwendet.

5.3.1 Echte Template Methode

Innerhalb der echten Template-Matching-Methode geht das Programm grundsätzlich von der Annahme aus: Ist der Hintergrund dunkel, dann muss der Abdruck hell sein und umgekehrt. Trifft das Programm auf ein am Randbereich dunkles Bild, wird das Bild invertiert. Es werden auch Überlegungen angestellt das Template mit einem Histogramm

an die richtigen Farbwerte anzupassen, das wird aber Aufgrund des viel größeren Erfolgs der Feature basierten Methode fallen gelassen.

Das Programm produziert zunächst Templates mit allen möglichen Durchmessern. Es kann ausgewählt werden ob ein Teil des Hintergrundes oder der Mittelwert des Hintergrundes die Umgebung des Kreises ausfüllen soll. Der Randbereich des Abdrucks im Template ist zunächst immer hell. Es handelt sich im den tonangebenden Bereich, er sollte hohe Werte haben um die Antworten zu maximieren. Einige Templates sind unten dargestellt um den Zusammenhang zu verdeutlichen (*Abbildung 32, Abbildung 33*).

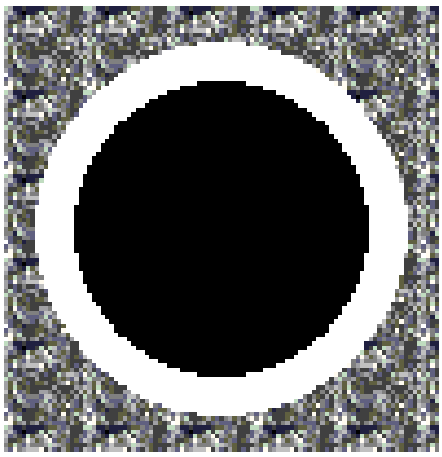


Abbildung 32: Template mit Textur-Hintergrund

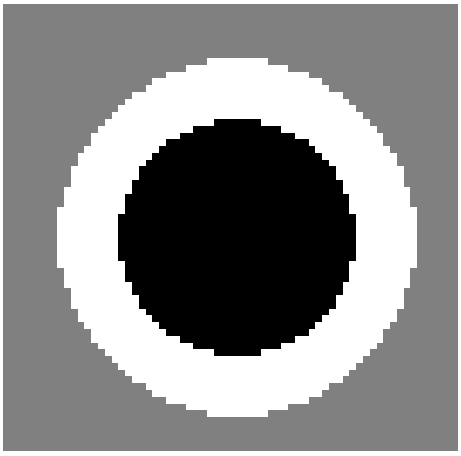


Abbildung 33: Template mit Mittelwert Hintergrund

Ist das Programm mit der Template Erzeugung fertig invertiert es gegebenenfalls das Bild, wenn es aufgrund einer hellen Umgebung einen dunklen Abdruck erwartet. Der Abdruck sollte dann durch die Umkehrung der Graustufen hell werden und gut auf die Templates ansprechen. Tatsächlich kommt es auch auf eine gute Abstimmung des Templates an, je nachdem wie die Dicke des weißen Abdruckrands und die Größe des Außenbereichs gewählt sind sprechen die Templates verschieden gut auf die jeweiligen Bilder an. In nachfolgender *Abbildung 34* ist das eingezeichnet.

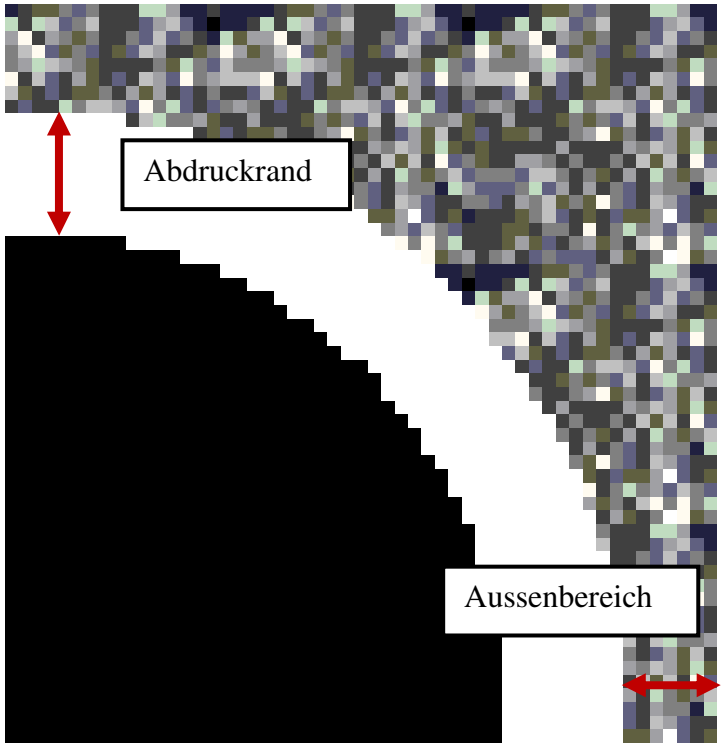


Abbildung 34: Variationsmöglichkeiten innerhalb des Templates

Der schwarze Bereich innerhalb der Templates sind Wildcards (*Abbildung 32*). Dieser Bereich wird weder bei der Berechnung des Template-Bild-Faltungsproduktes, noch bei der Berechnung des Normierungs-Faltungsproduktes berücksichtigt. Das bedeutet diese Werte sind immer Null und fallen bei der Summierung als Nullsummanden weg.

Das Matching des Templates geschieht nach der Methode des normierten Kokorellationskoeffizienten. Die generalisierte Formel dafür lautet wie folgt *Gl. 33*.

$$R(x,y) = \frac{\sum_{x^*,y^*}\left(T^*(x^*,y^*)\cdot I^*(x+x^*,y+y^*)\right)}{\sqrt{\sum_{x^*,y^*} T^*(x^*,y^*)^2 \cdot \sum_{x^*,y^*} I^*(x+x^*,y+y^*)^2}}$$

Gl. 33

Das Ergebnis in *Gl. 33,* also $R(x,y)$, berechnet sich immer durch Summation über alle Pixel des Templates, sie werden mit x^*, y^* bezeichnet. Im Gleichklang findet eine Multiplikation mit den darunterliegenden Bildpixeln statt, was ist wiederum die Kernoperation des Faltungsproduktes widerspiegelt.

Um die *Gl. 33* zu interpretieren, muss zunächst überlegen werden, wann $R(x,y)$ den Wert 1 annehmen kann. Angenommen es liegt ein Template vor, welches zuvor aus einem Bild ausgeschnitten wurde, dazu *Abbildung 35*.

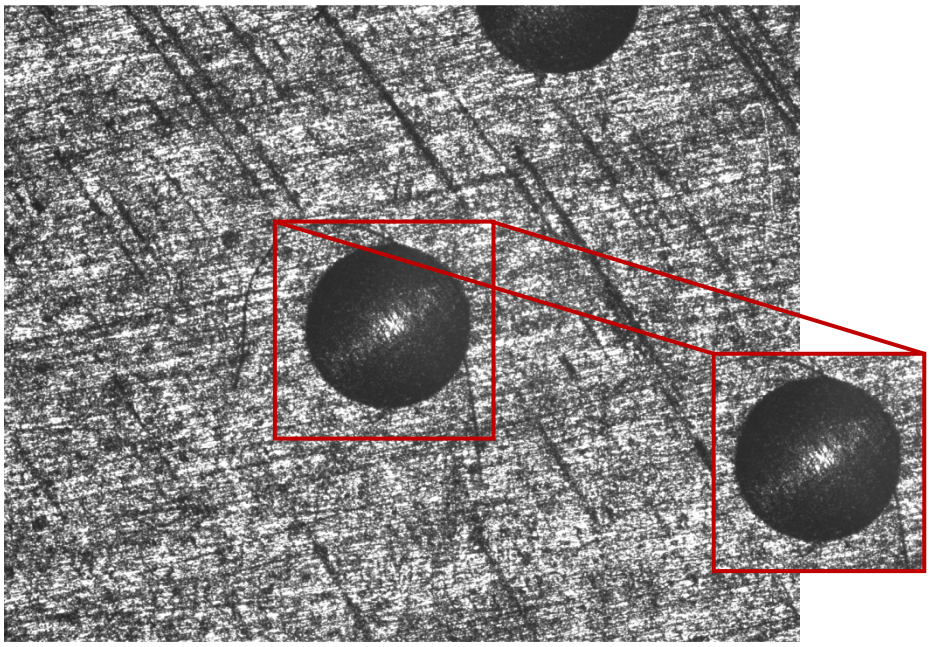


Abbildung 35: Beispiel für ein Template, dass aus einem Bild ausgeschnitten wird

Wird das Template aus *Abbildung 35* mit dem Bild aus *Abbildung 35* gematcht, so wird der größte $R(x,y)$-Wert an der Stelle des Abdrucks auftreten. In dem Fall, des vorherigen Ausschneidens des Abdrucks an genau jener Stelle sollte sich dort sogar $R(x,y) = 1$ ergeben, da für diesen Spezialfall gilt $T^*(x^*,y^*) = I^*(x+x^*,y+y^*)$, sprich die unterliegenden Bildpixel stimmen exakt mit dem Template überein, verwandelt sich der Ausdruck $\sum_{x^*,y^*}\left(T^*(x^*,y^*)\cdot I^*(x+x^*,y+y^*)\right)$ dann mehr oder weniger in $\sum_{x^*,y^*}(T^*(x^*,y^*)^2)$. Der Term $\sqrt{\sum_{x^*,y^*}T^*(x^*,y^*)^2\cdot\sum_{x^*,y^*}I^*(x+x^*,y+y^*)^2}$ lässt sich zunächst in die Formel $\sqrt{\sum_{x^*,y^*}T^*(x^*,y^*)^2\cdot\sum_{x^*,y^*}T^*(x^*,y^*)^2}$ transformieren und in weiterer Folge zu $\sum_{x^*,y^*}T^*(x^*,y^*)^2$ umschreiben. Nun stimmen Nenner und Zähler des Bruchs in *Gl. 33* überein und damit folgt $R(x,y) = 1$.

Die Wurzel im Nenner von *Gl. 33* kann also als geometrisches Mittel zwischen der maximalen Antwort, die erzielt werden kann, wenn das Template an der Stelle (x,y) exakt mit dem Bild übereinstimmt: $\sum_{x^*,y^*}I^*(x+x^*,y+y^*)^2$ und der maximalen Summe die erreicht wird falls das unterliegende Bild exakt mit dem Template zusammen passt, das ist $\sum_{x^*,y^*}T^*(x^*,y^*)^2$, interpretiert werden.

Diese Normierung ist anschaulich klar, denn lässt man ein Template mit einer Stelle im Bild matchen, die aus Bildgründen niedrige Werte hat, da es sich um einen dunklen Bereich handelt, kann im Vergleich zu einem Matching an einer hellen Stelle mit naturgemäß hohen Werten nicht durch den gleichen Normierungswert geteilt werden, da

sonst das Matching im dunklen Bereich benachteiligt werden würde. Wobei mit Normierungswert der Nenner in *Gl. 33* gemeint ist. Somit gleicht der Nenner von *Gl. 33*, einen hellen und dunklen Bildbereich auf natürliche Weise aus und korrigiert sich nach oben bzw. nach unten.

Die Werte des Templates selbst müssen auch in den Normierungsfaktor einfließen, denn ein helles Template ist quasi im Stande viel höhere Werte zu produzieren, muss also durch größere Zahlen dividiert werden um auf den Bereich $[0,1]$ beschränkt werden zu können.

Die Methode des normierten Kokorrelationskoeffizienten, lässt sich gänzlich auf die Berechnung von Faltungsprodukten reduzieren und kann in Matlab also relativ effizient realisiert werden. Eine Umwälzung in den Frequenzraum bringt theoretisch vielleicht einen Nutzen, ist aber bei schlecht überlegter Implementierung in Matlab sogar langsamer. Um sich eine wirklich durchdachte Implementierung zu überlegen sollte genau untersucht werden, ab welchen Bildgrößen die Hin- und Rücktransformation gegen die herkömmliche Berechnung der Faltung in der Ortsdomäne rentabel ist.

Innerhalb des Programms wird *Gl. 33* etwas verändert. Zunächst ist die Berechnung des Nenners in *Gl. 33* sehr aufwendig, da die Wurzelberechnung auf iterative Verfahren zurückgreifen muss. Wird *Gl. 33* quadriert, dann ändert sich die Wertespanne von $R(x,y)$ nicht und bleibt im Intervall $[0,1]$ liegen.

$R(x,y) = \frac{\left[\sum_{x^*,y^*}\left(T^*(x^*,y^*)\cdot I^*(x+x^*,y+y^*)\right)\right]^2}{\sum_{x^*,y^*} T^*(x^*,y^*)^2 \cdot \sum_{x^*,y^*} I^*(x+x^*,y+y^*)^2}$	**Gl. 34**

Der einzige Nebeneffekt ist eine kleine Veränderung der Korellationswerte. Alle $R(x,y)$ werden etwas kleiner und schwache Übereinstimmungen stärker benachteiligt als starke. Der Zusammenhang ist direkt aus dem Vergleich der Parabel $p(x) = x^2$ und der Geraden $g(x) = x$ im Intervall $[0,1]$ abzuleiten (*Abbildung 36*).

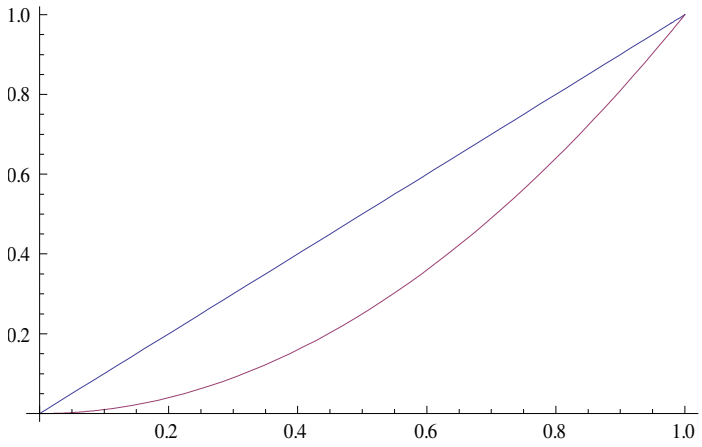


Abbildung 36: Verlauf eines Polynoms zweiten Grades mit alleinigem quadratischen Koeffizienten ungleich Null aber Eins gegen den Verlauf eines Polynoms ersten Grades mit nur dem linearen Koeffizienten ungleich Null aber Eins

Die Berechnung des Nenners und des Zählers in *Gl. 34* erfolgt getrennt. Tatsächlich werden Nenner- und Zähler-Matrizen berechnet. Es werden also alle $R(x,y)$ pseudo gleichzeitig ermittelt. Die Ergebnismatrix für (R_{ij}) entsteht durch elementweise Division von Nenner- durch Zählermatrix. Das Ergebnis der Faltung ist eine verkleinerte Matrix. Am Rand der Bildmatrix hat das Template keinen Platz mehr und die Berechnung kann nicht mehr sinnvoll durchgeführt werden. Es besteht natürlich die Möglichkeit die Bildmatrix mit einem Rahmen jeglicher Art zu versehen und dann auch am Rand ein Matching durchzuführen, das ist hier aber nicht nötig da das Objekt im Inneren des Bildes und als Ganzes vorhanden vorausgesetzt wird. Nachfolgend werden nochmals die Überdeckung des Templates mit der Bildmatrix gezeigt und der nichtberechenbare Bereich dargestellt (*Abbildung 37*).

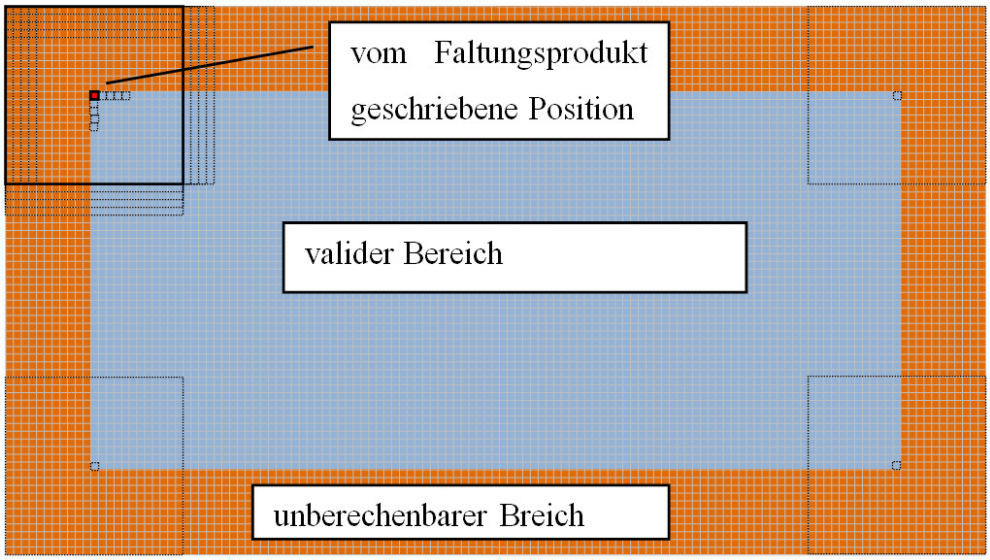


Abbildung 37: valider Bereich des Faltungsproduktes

Innerhalb dieser Implementierung wird nach der Division der Nenner- und Zähler-Matrix der nicht berechenbare Bereich mit Nullen aufgefüllt. Eine vorherige Polsterung mit Nullen resultiert in einer Division durch Null, was in Matlab im Allgemeinen keine Exception wirft, sondern das Ergebnis *NaN* oder *Inf* liefert.

Für eine Bildmatrix der Größe $m \times n$ und ein Template der Größe $t \times t$ ergibt sich eine Ergebnismatrix der Größe $(m - t - 1) \times (n - t - 1)$. Die entsprechenden Größendifferenzen können dann im Endergebnis mit Nullen gefüllt werden.

Die Zähler-Matrix ergibt sich aus der Bildung des Faltungsproduktes Templatematrix-Bildmatrix und anschließendes elementweises Quadrieren der Ergebnismatrix: $(T * I)^{.2}$. Die Templates sind immer quadratisch und haben eine ungerade Seitenlänge.

Um den Nenner aus *Gl. 34*. zu berechnen kann auch eine Faltungsoperation ausgeführt werden. Die Summe der Template-Werte $\sum_{x^*,y^*} T^*(x^*, y^*)^2$ ist für alle möglichen Mittelunkte konstant. Diese Summe kann also einmal berechnet werden und in eine Matrix der Größe des Templates geschrieben werden. Diese Matrix hat dann folgende Gestalt: $A = \left(a_{ij} = \sum_{x^*,y^*} T^*(x^*, y^*)^2\right)_{\substack{1 \le i \le TemplateSize \\ 1 \le j \le TemplateSize}}$. Falls Wildcards verwendet werden, müssen diese Positionen in dieser Matrix auf Null gestellt werden. Anschließend kann die Matrix A mit der elementweise quadrierten Bildmatrix I^* gefaltet werden: $A * I^{*.2}$.

Nenner-Matrix und Zähler-Matrix bzw. (R_{ij}) ergeben sich also, wie in *Gl. 35* angeführt.

$M_{Zähler} = (T * I)^{.2}$ $A = \left(a_{ij} = \sum_{x^*,y^*} T^*(x^*, y^*)^2 \right)_{\substack{1 \le i \le TemplateSize \\ 1 \le j \le TemplateSize}}$ $M_{Nenner} = A * I^{*.2}$ $R = \frac{M_{Zähler}}{M_{Nenner}}$	**Gl. 35**

Die Ergebnismatrix (R_{ij}) wird nun noch an den Stellen des unberechenbaren Bereichs mit Nullen erweitert und erreicht wider die Größe des Bildes.

Das Programm findet das passendste Template auf einfache Weise. Das größte globale Maximum jedes Faltungsproduktes der jeweiligen Template-Größen wird als Fundstelle

interpretiert, und der Kreis dorthin gezeichnet. Nachfolgend sind zwei Faltungsproduktergebnisse als drei-dimensionale Plotte aufgetragen, daneben sind auch die Templates und das Bild im Original-Verhältnis zu sehen.

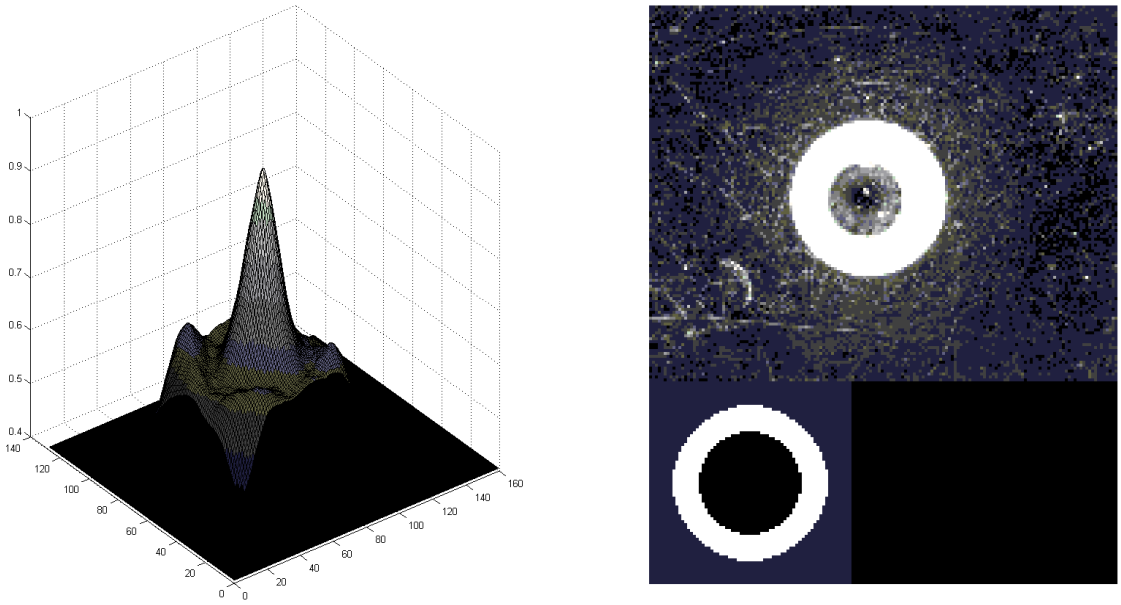


Abbildung 38: Template passt an einer Stelle gut - Es kann klar und deutlich ein scharfes globales Maximum gesehen werden.

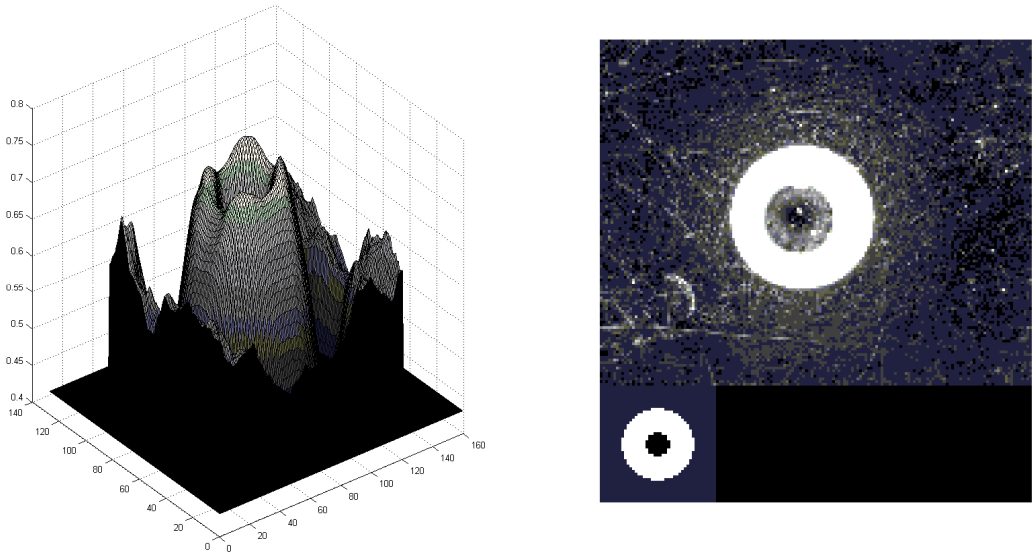


Abbildung 39: Template passt an allen Stellen schlecht - Ein eindeutiges globales Maximum ist nicht zu erkennen.

5.3.2 Feature Matching

Die an die Verkehrszeichenerkennung in manchen Autos angelegte Methode funktioniert besser als das Matchen eines echten Templates mit angepassten Grauwerten und Hintergründen. Als feature werden die Kanten des zu matchenden Bildes verwendet. Die Berechnung des Edge-Bildes wird mit dem Canny-Algorithmus durchgeführt.

Die Templates dieser Methode haben sehr einfache Gestalt. In eine Matrix wird ein Pixelkreis gezeichnet. Die Matrix hat nur an Stellen des Kreises von Null verschiedene Werte. Eine Kreisdickeneinstellung von 2 oder etwas mehr ist bei diesen Templates wichtig, da sonst ein etwas deformierter Kreis nicht gut erkannt wird. Nachfolgende *Tabelle 6* und *Abbildung 40* zeigen die Kreismatrix mit den jeweiligen Werten und das Template als Schwarzweißbild.

0	0	0	0	0	2	2	2	0	0	0	0	0
0	0	0	2	2	2	2	2	2	2	0	0	0
0	0	2	2	2	2	2	2	2	2	2	0	0
0	2	2	2	2	0	0	0	2	2	2	2	0
0	2	2	2	0	0	0	0	0	2	2	2	0
2	2	2	0	0	0	0	0	0	0	2	2	2
2	2	2	0	0	0	0	0	0	0	2	2	2
2	2	2	0	0	0	0	0	0	0	2	2	2
0	2	2	2	0	0	0	0	0	2	2	2	0
0	2	2	2	2	0	0	0	2	2	2	2	0
0	0	2	2	2	2	2	2	2	2	2	0	0
0	0	0	2	2	2	2	2	2	2	0	0	0
0	0	0	0	0	2	2	2	0	0	0	0	0

Tabelle 6: Matrix des Kreistemplates mit Werten

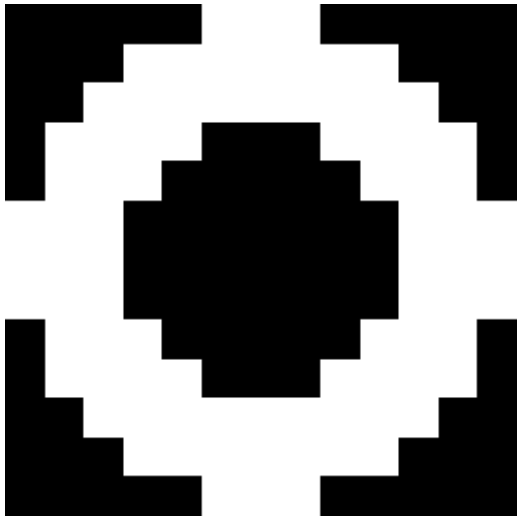


Abbildung 40: Kreistemplate als Schwarzweißbild

Die obigen Templates sind kleine Vertreter der verwendeten Exemplare. Alle Nullen in *Tabelle 6* sind Wildcards und gehen in die Berechnung nicht ein. Dieser Tatsache wird bei der Berechnung des Faltungsproduktes der Nenner-Matrix aus *Gl. 34* speziell Rechnung getragen. $\sum_{x^*,y^*} T^*(x^*,y^*)^2$ ist für das Template in *Tabelle 6* 336. Es muss also eine Matrix mit den Werten 336 und der Template Größe erstellt werden, wobei die Wildcard-Bereiche auszublenden sind. Werden die Wildcard-Bereiche nicht ausgeblendet, werden

die entsprechenden Werte des Bildes bei Berechnung der Faltung berechnet und der Nenner von *Gl. 34* wird zu groß. In *Gl. 36* wird das nochmals dargestellt, wobei A die Matrix A aus *Gl. 35* ist.

$$A = \begin{bmatrix} 336 & \cdots & 336 \\ \vdots & \ddots & \vdots \\ 336 & \cdots & 336 \end{bmatrix}$$

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$A^{'} = A .* (\sim W)$$

$$A^{'} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 336 & 336 & 336 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 336 & 336 & 336 & 336 & 336 & 336 & 336 & 0 & 0 & 0 \\ 0 & 0 & 336 & 336 & 336 & 336 & 336 & 336 & 336 & 336 & 336 & 0 & 0 \\ 0 & 336 & 336 & 336 & 336 & 0 & 0 & 0 & 336 & 336 & 336 & 336 & 0 \\ 0 & 336 & 336 & 336 & 0 & 0 & 0 & 0 & 0 & 336 & 336 & 336 & 0 \\ 336 & 336 & 336 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 336 & 336 & 336 \\ 336 & 336 & 336 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 336 & 336 & 336 \\ 336 & 336 & 336 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 336 & 336 & 336 \\ 0 & 336 & 336 & 336 & 0 & 0 & 0 & 0 & 0 & 336 & 336 & 336 & 0 \\ 0 & 336 & 336 & 336 & 336 & 0 & 0 & 0 & 336 & 336 & 336 & 336 & 0 \\ 0 & 0 & 336 & 336 & 336 & 336 & 336 & 336 & 336 & 336 & 336 & 0 & 0 \\ 0 & 0 & 0 & 336 & 336 & 336 & 336 & 336 & 336 & 336 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 336 & 336 & 336 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Gl. 36

Die Tilde in *Gl. 36* steht für die Negation der logischen Elemente der Matrix und $.*$ für die elementweise Multiplikation. Das Ergebnis A' wird nun mit dem elementweise quadrierten Bild nach *Gl. 35* gefaltet, die Wildcard-Positionen fallen also Nullsummanden auch bei der Berechnung des Nenners heraus.

Abschließend sind noch drei-dimensionale Plots der Ergebnismatrizen für das feature-Matching für ein gut und ein schlecht passendes Template abgebildet (*Abbildung 41, Abbildung 42*).

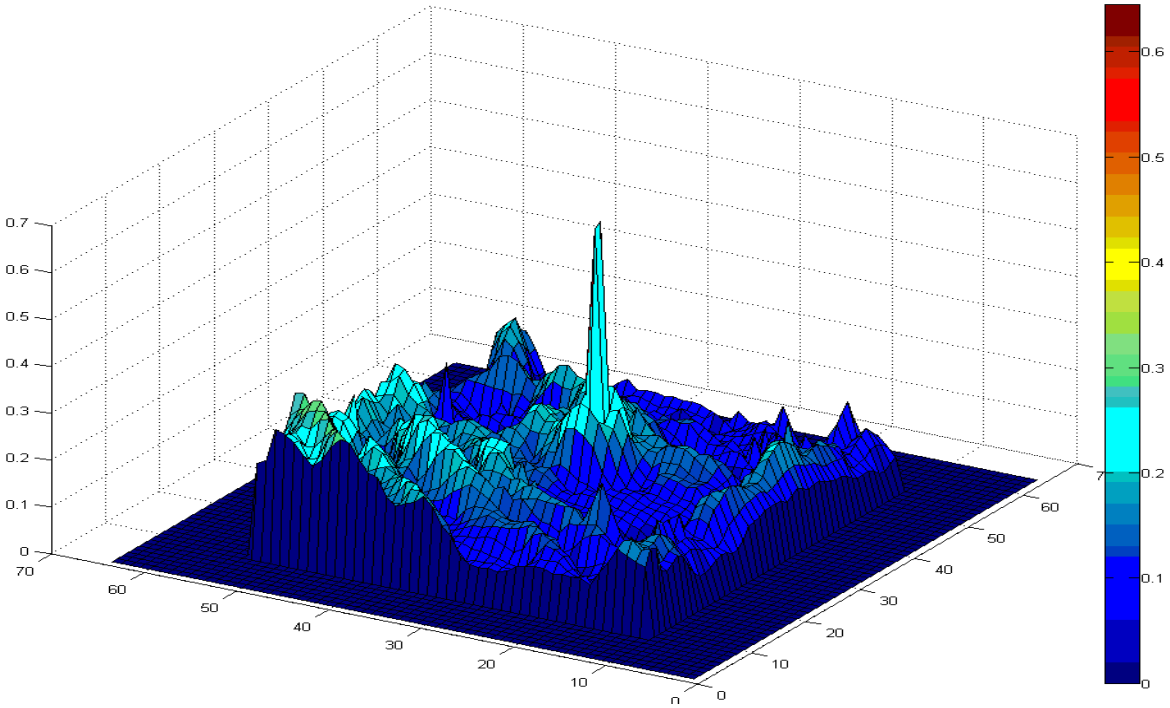


Abbildung 41: Kokorelationsmatrix für ein gut passendes Template

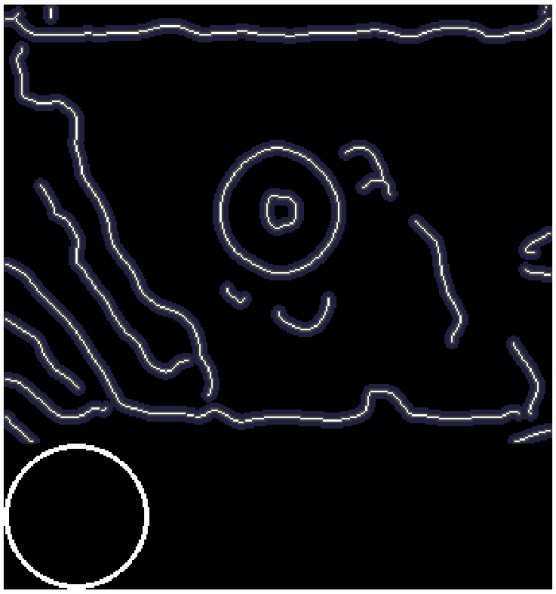


Abbildung 42: Kantenbild und dazugehöriges gut passendes Template

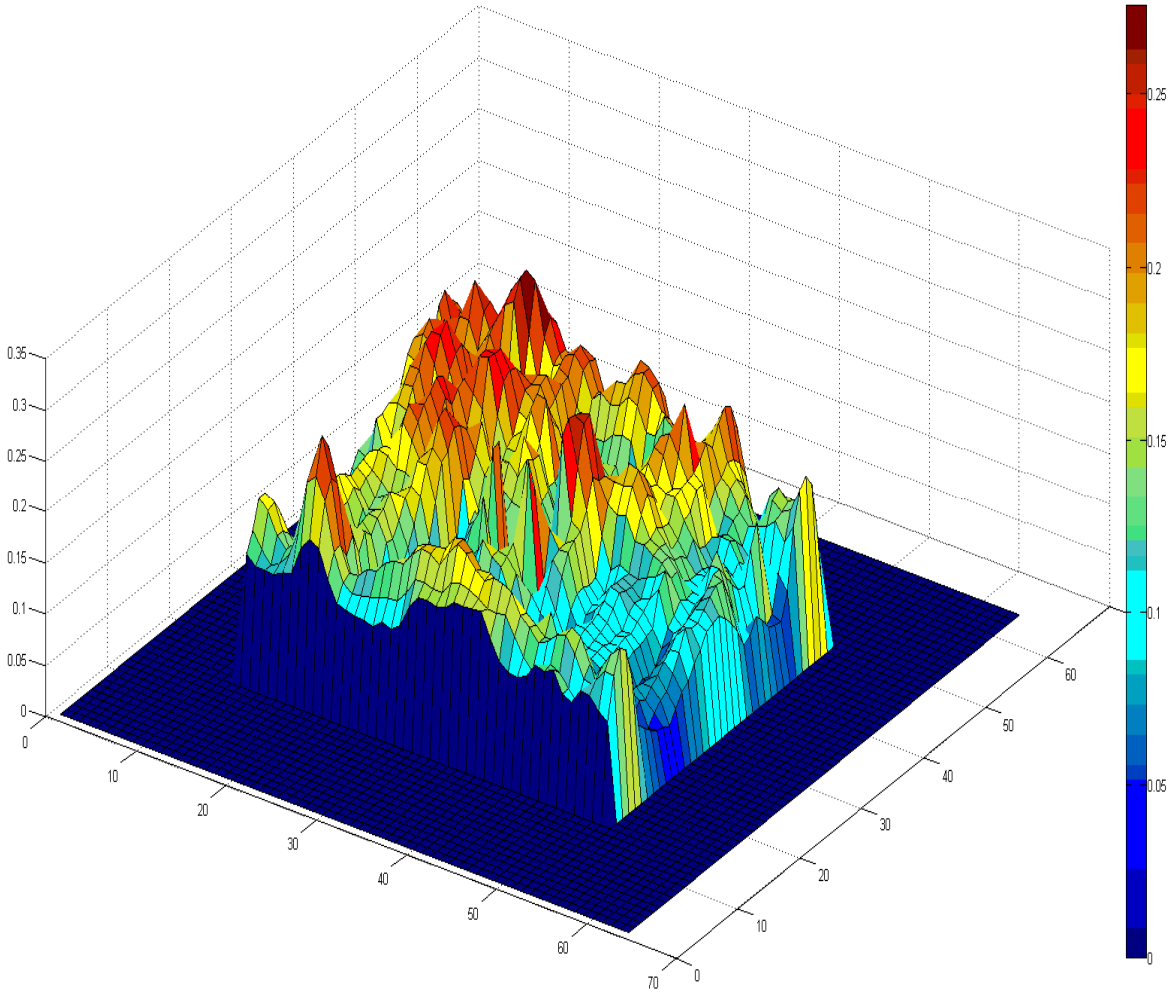


Abbildung 43: Korelationskoeffizientenmatrix für ein nirgends passendes Template

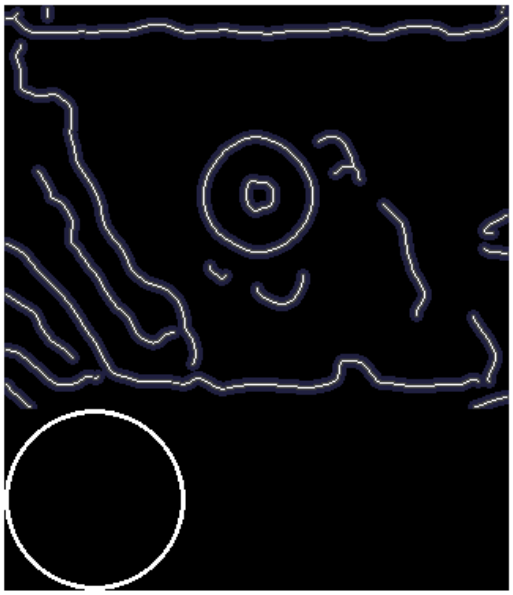


Abbildung 44: Dieses Kreistemplate passt nirgendwo im Bild gut

In *Abbildung 43* ist ein eindeutiges globales Maximum zu finden. Hingegen gibt es in *Abbildung 44* zwar einige lokale Maxima, aber keinen echten Peak der alle anderen übersteigt. Die Suche des besten Mittelpunktes ist wiederum eine Suche des größten Wertes in allen Faltungsproduktergebnissen.

Abschließend sollte noch erwähnt werden, dass im Falle des Feature-Template-Matching auch das Bild selbst Wildcards hat. Alle Bereiche in dem sich das Template mit gar keiner Kante überschneidet sind unwichtig und werden bei der Berechnung sofort auf null gestellt.

Das Programm berechnet die Korrelationsmatrix für jede Kreis-Template-Größe. Anschließend werden die Ergebnismatrizen gestapelt und das Maximum in einer dreidimensionalen Matrix gesucht um den Mittelpunkt des Kreises, sowie dessen Radius zu erhalten. Die Kreis-Templates von verschiedenem Durchmesser können durch das Programm einmal erzeugt und dann abgespeichert werden, somit muss die Berechnung des Pixelkreises nicht immer durchgeführt werden, was die Laufzeit verkürzt.

Falls konzentrische Kreise auftreten wird immer der größte Kreis gleicher Intensität herangezogen.

5.4 Kernfunktion des Integro-Differential-Operator-Programms

Die Idee hinter dem Integro-Differential-Operator stammt von John Daugman. Er entwickelte das Verfahren für die Einschränkung des Irisbereichs um eine Muster-Erkennung durchzuführen.

Ähnlich wie der Sobel-Operator findet das Programm Kanten indem es die numerische Ableitung des Bildes berechnet. Der wesentliche Unterschied ist nun, dass nur Kanten auf Kreisformen in tangentiale Richtung untersucht werden. Das bedeutet, es werden alle kreisrunden Kantenverläufe für jeden möglichen Kreismittelpunkt im Bild betrachtet. Liegt an irgendeinem konzentrischen Kreis des entsprechend zu untersuchenden Mittelpunktes ein Maximum im Gradienten vor, so muss es sich um einen tatsächlichen Kreis im Bild handeln.

Kanten die parallel zur radialen Richtung liegen werden in der Berechnung nicht berücksichtigt. Ein ähnliches Ergebnis kann man erreichen, wenn man in einem einfachen Kanten-Bild für einen gegebenen Mittelpunkt alle Kantenrichtungen auf einen

Radiusvektor projiziert. Gradientenvektoren in tangentialer Richtung haben dann keine entsprechende Komponente und fallen weg.

Eine Projektion des Spalten- und Zeilengradienten in radiale Richtung kann in Matlab relativ einfach realisiert werden. Um den Vorgang der Projektion zu zeigen wird ein Testbild verwendet. Es wird a prori von einem bekannten Mittelpunkt ausgegangen. Die Kreisform ist nicht bekannt, aber es liegt eine scharfe kreisrunde und eine scharfe durch den Mittelpunkt der kreisrunden Kante laufende Kante in radiale Richtung vor. Das Testbild (*Abbildung 45*) hat einfache gestellt. Der Mittelpunkt sei in diesem Fall vorgegeben und wird in das Bild eingezeichnet.

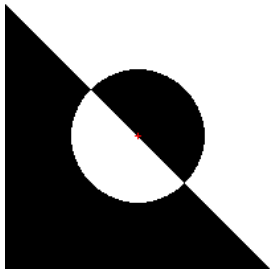


Abbildung 45: Das rote Kreuz markiert den Mittelpunkt des Kreises

Mit einem einfachen Test wird nun gezeigt, dass der Daugman-Integro-Differential-Operator auch im Stande ist einen Kreis der obigen Art zu finden. Der Punkt ist nämlich, dass sich der Daugman-Integro-Differential-Operator für die diagonal, also in radiale Richtung verlaufenden Kanten gar nicht interessiert. Natürlich hat eine Kante, die nicht durch den angenommenen Mittelpunkt verläuft auch eine gewisse radiale Komponente für irgendeinen Kreis durch den rot markierten Mittelpunkt in *Abbildung 45*, aber diese Kante liefert, wenn sie nicht kreisrund um den Mittelpunkt verläuft keine große Antwort.

Zunächst wird der numerische Gradient in Spalten und Zeilenrichtung berechnet. Das wird mit der Matlab-Funktion *diff* erreicht. Um diesen Vorgang tatsächlich in Matlab zu implementieren muss noch beachtet werden, dass sich möglicherweise die Gradientenwerte, in den jeweiligen Spalten- und Zeilen-Gradienten-Matrizen nicht an den gleichen Positionen befinden, das kann aber leicht durch eine Shifting-Operation behoben werden.

Für jeden Punkt des Bildes in *Abbildung 45* wird gedanklich eine Gerade zum Mittelpunkt des Kreises gezogen. Es wird dann davon ausgegangen, diese Gerade wäre der Radius eines Kreises. In Folge dessen kann durch jeden Bildpunkt ein entsprechender Kreis mit dem angenommen Mittelpunkt konstruiert werden. Ein Steigungsdreieck für diese Gerade

ergibt sich leicht aus den Offset-Koordinaten in Spalten und Zeilenrichtung zum Mittelpunkt. Mit diesem Steigungsdreieck kann ein Radiusnormalvektor für den angenommen Kreis aufgestellt werden. Dazu wird ein willkürlicher Punkt aus *Abbildung 45* genommen und in *Abbildung 46* als Kreispunkt eingetragen.

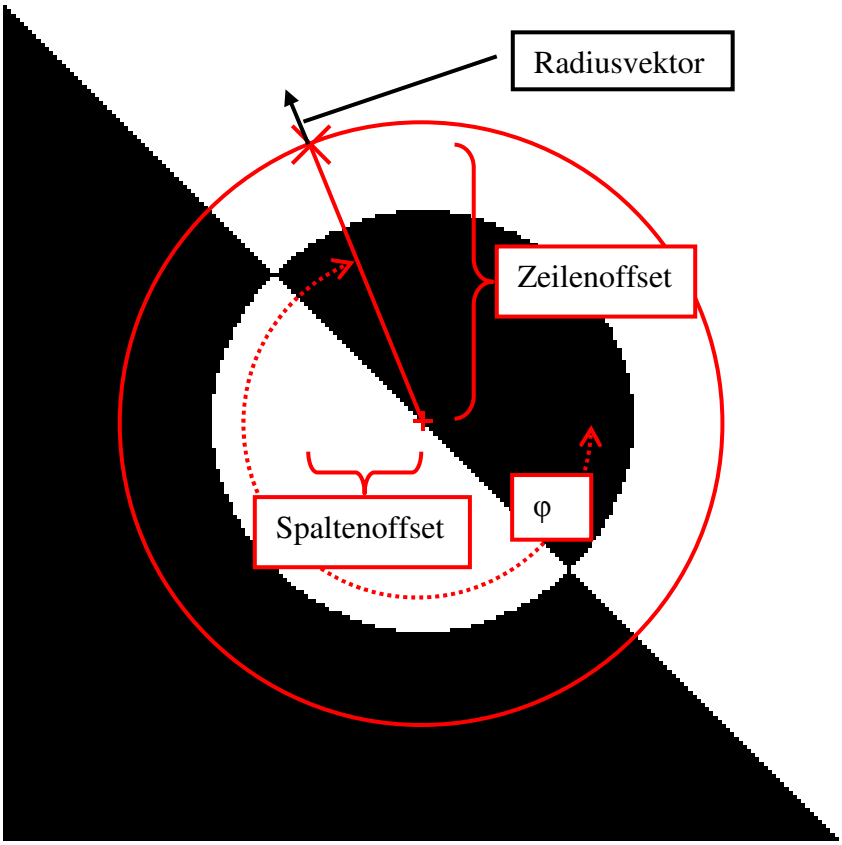


Abbildung 46: Kreis durch den angenommenen Punkt

Der Radiusnormalvektor lässt sich nun einfach ermitteln. Dazu wird der Radius einfach durch quadratisches Addieren der Spalten- und Zeilenoffsets zum Mittelpunkt und anschließendes Wurzelziehen berechnet. Die Spaltenkoordinate ergibt sich dann durch Division von Spaltenoffset zum Mittelpunkt durch den Radius und die Zeilenkoordinate ist entsprechend die Division des Zeilenoffsets durch den Radiusbetrag. Das ist nichts anderes als die Sinus und Kosinus Werte des eingezeichneten Winkel φ. In Matlab lässt sich also der radiale Richtungsvektor quasi-parallel mit Matrizen berechnen. Der Radiusrichtungsvektor ist also durch *Gl. 37* wie bereits erläutert gegeben.

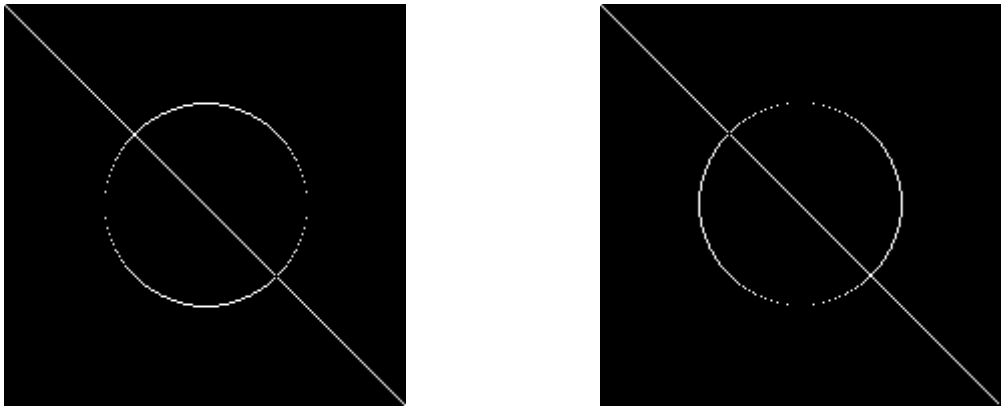
$$r = (\sin\varphi\,, \cos\varphi)$$

$$r = \left(\frac{Zeilenoffset}{\sqrt{Zeilenoffset^2 + Spaltenoffset^2}}, \frac{Spaltenoffset}{\sqrt{Zeilenoffset^2 + Spaltenoffset^2}}\right)$$

Gl. 37

In Matlab kann die Spalten und Zeilenkomponente des Radialeinheitsvektors jedes beliebigen Punktes in zwei Matrizen gespeichert werden. Die Ausführung eines Skalarproduktes zwischen Einheitsvektos und Punktvektors entspricht der Projektion dieses Vektors in die Richtung des Einheitsvektors.

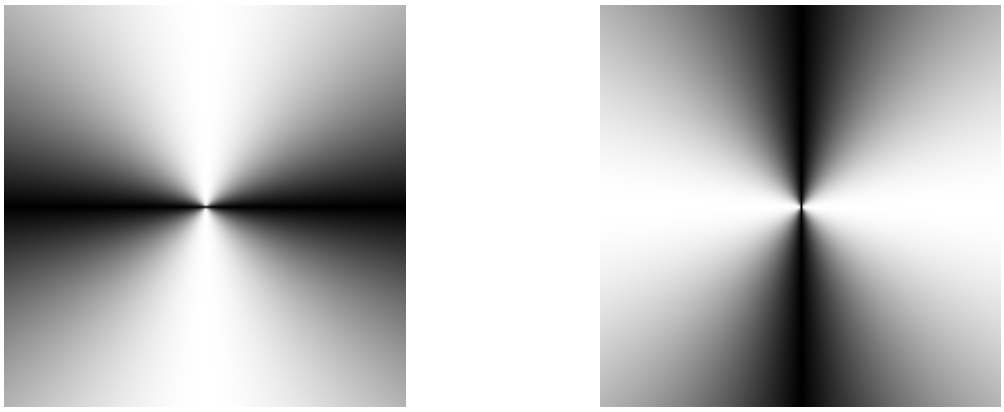
Die Gradientenwerte können auch einzeln in einer Matrix für die Zeilenkomponenten und eine Matrix für die Spaltenkomponenten gespeichert werden. Diese Matrizen sind nachfolgend dargestellt (*Abbildung 47*). Es ist zu beachten, dass hier der Absolutbetrag des Gradienten verwendet wird.



(a) **(b)**

Abbildung 47: Gradienten in (a) Spalten- und (b) Zeilenrichtung

Die Absolutbeträge des Radiusnormalvektors für jeden Punkt lassen sich wie erwähnt auch in zwei Matrizen speichern und darstellen. Diese Matrizen sind in *Abbildung 48* dargestellt. Es ist darauf zu achten, dass in diesen Bildern hohe negative Werte von hohen positiven Werten nicht unterschieden werden können.



(a) **(b)**

Abbildung 48: Radiusnormalvektor Spalten- und Zeilenrichtung

In Matlab kann das innere Produkt für alle Punkte gleichzeitig ausgeführt werden. Dies geschieht in dem die Matrizen aus den *Abbildung 47* und *Abbildung 48* wie folgt elementweise multipliziert und addiert werden (*Abbildung 49*).

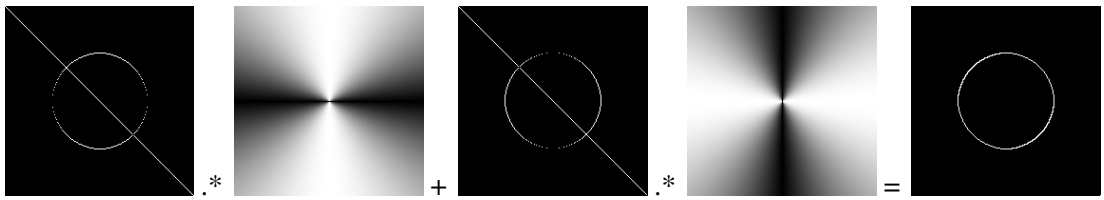


Abbildung 49: Ergebnis der Projektion des Gradienten auf die radiale Richtung

Das zeigt nun, dass sich eine Projektion aller Gradientenrichtungen auf eine radiale Richtung in Matlab sehr leicht realisieren lässt. Zusätzlich wird auch plausibel, dass der Daugman-Integrodifferential-Operator praktisch gegen Kantenverläufe parallel, oder mit einem hohen Anteil parallel zur radialen Richtung, immun ist und sich von diesen Kanten nicht beeinflussen lässt. Natürlich kann es vorkommen, dass eine Kante nicht zum Kreis gehört und trotzdem genau tangential verläuft und somit auch ein Maximum in der Radialprojektion erreicht, da der Daugman-Integrodifferential-Operator aber auch kreisrunde Pfad-Integrale verwendet hat das keine großen negativen Auswirkungen. Zusammengefasst ist der Daugman-Integrodifferential-Operator also in der Lage schwere Kreisformen, wie in *Abbildung 45* zu finden.

Zu der Radialprojektion in Matlab sei noch angemerkt, dass eine Gerade, die nicht durch den Kreismittelpunkt geht, natürlich auch gewisse Kanten-Komponenten in Radial-Richtung aufweist und dadurch im projizierten Kantenbild nicht ganz verschwindet.

Die von John Daugman publizierte Formel zu seinem Verfahren der Kreiserkennung hat folgende Gestalt (*Gl. 38*).

$$max(r, x_0, y_0) \left| G_\sigma(r) * \frac{\partial}{\partial r} \int_{r_{min}}^{r_{max}} \oint \frac{I\big(x(s), y(s)\big)}{2\pi r} ds\, dr \right| \qquad \textbf{Gl. 38}$$

Ein Teil dieser Formel kann auf die Berechnung von Faltungsprodukten zurückgeführt werden. In Matlab kann das Pfad-Integral in *Gl. 38* für alle möglichen Bildpunkte und einen gegeben Radius quasi parallel berechnet werden. Dazu wird eine Faltung zwischen Bild und einem entsprechenden Template (*Abbildung 50*) ausgeführt.

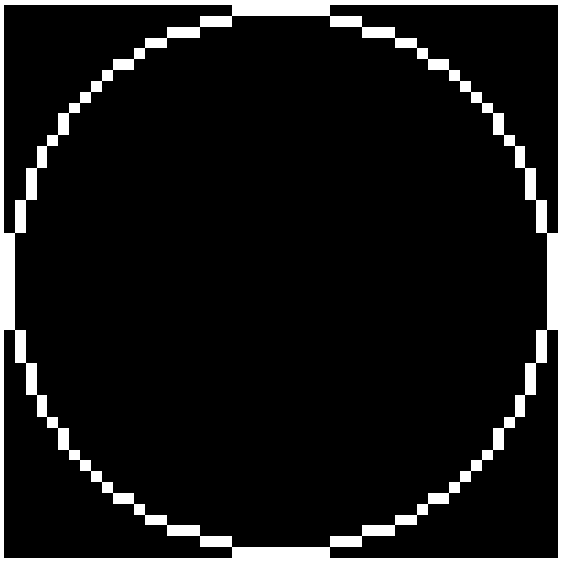


Abbildung 50: Pfad-Integral Template

Das Template in *Abbildung 50* ist ein binäres Template. An den schwarzen Stellen sind Nullen und an den weißen Stellen Einsen gesetzt. Das Template hat eine ungerade Seitenlänge und ist für jeden Radius quadratisch. Somit hat es auch einen Pixel als Mittelpunkt. Innerhalb des Faltungsproduktes werden dann alle Werte auf dem Pfadintgral addiert und an den entsprechenden möglichen Mittelpunkt in der Ergebnismatrix geschrieben.

Es ist also möglich für den selben Radius alle Pfadintegrale für jeden möglichen Mittelpunkt, dass sind vorzugsweise alle Pixel des Bildes, pseudo parallel zu berechnen. Es ist hier von „pseudo“ oder „quasi“ parallel die Rede, da die Art der Implementierung des Faltungsproduktes in Matlab unbekannt ist, und nicht gesagt werden kann welche Operationen in modernen Prozessoren tatsächlich gleichzeitig ablaufen.

Die Pfadintegrale über jeden Radius jedes Mittelpunktes werden in einer drei dimensionalen Matrix gespeichert. Dadurch besteht die Möglichkeit alle zu differenzierenden Daten in Matlab relativ speicherschonend abzulegen. In nachfolgender Grafik wird das nochmals verdeutlicht (*Abbildung 51*).

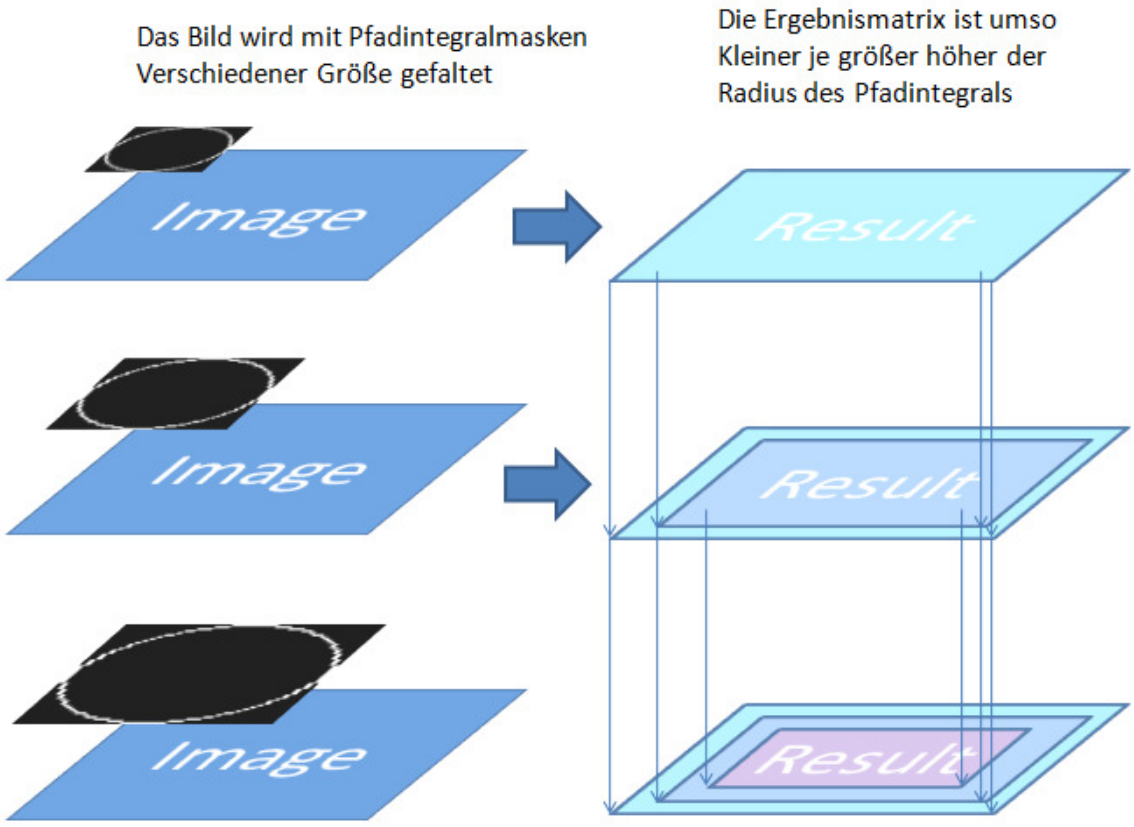


Abbildung 51: Das Bild wird mit Pfadintegralmasken verschiedener Größe gefaltet

Die Ergebnismatrizen nehmen, wie es in *Abbildung 51* dargestellt ist, mit der Größe der Pfadintegralmaske ab. Sie werden deshalb mit den entsprechenden Werten der vorherigen kleineren Maske am Rand aufgefüllt. Alle Ergebnismatrizen haben dann die gleiche Größe und können in einer einzigen drei-dimensionalen Matrix abgelegt werden. Innerhalb dieser Matrix liegt dann gleich wie in *Abbildung 51* das Ergebnis der Faltung mit der kleinsten Maske ganz oben. Durch das Auffüllen mit konstanten Werten an der Seite wird ein konstantes Null-Ergebnis nach der numerischen Ableitung sicher gestellt. Da die Absolutwerte zur Maximumsuche herangezogen werden könnten starke negative Werte zu einer Verfälschung des Ergebnisses führen.

Innerhalb des drei-dimensionalen Stapels der Pfadintegralergebnisse wird zunächst entlang der Höhe abgeleitet und anschließend mit einem gausschen Lowpass-Filter geglättet. Obwohl Daugman den glättenden Gaussfilter mit dem Radius variiert wird er in dieser Implementierung konstant gelassen. In nachfolgender *Abbildung 52* ist nochmals die Bearbeitung der Pfadintegralergebnisse für die jeweiligen Mittelpunkte in der dreidimensionalen Matrix aufgezeichnet.

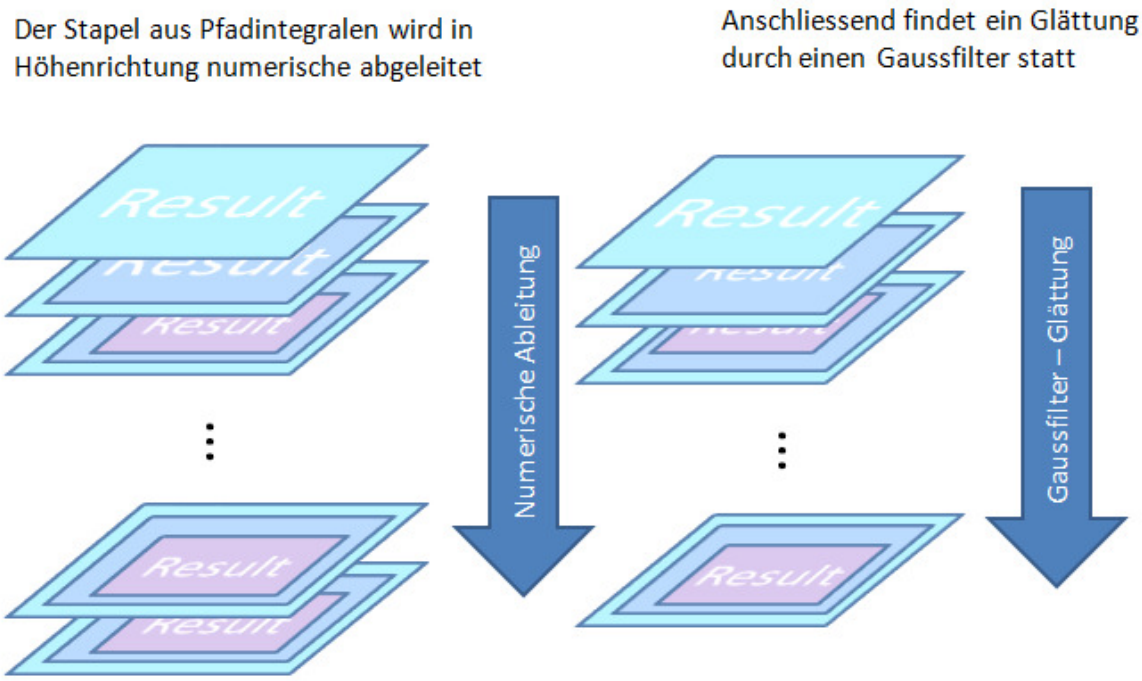


Abbildung 52: Differntiation der drei dimensionalen Matrix und anschließende Glättung

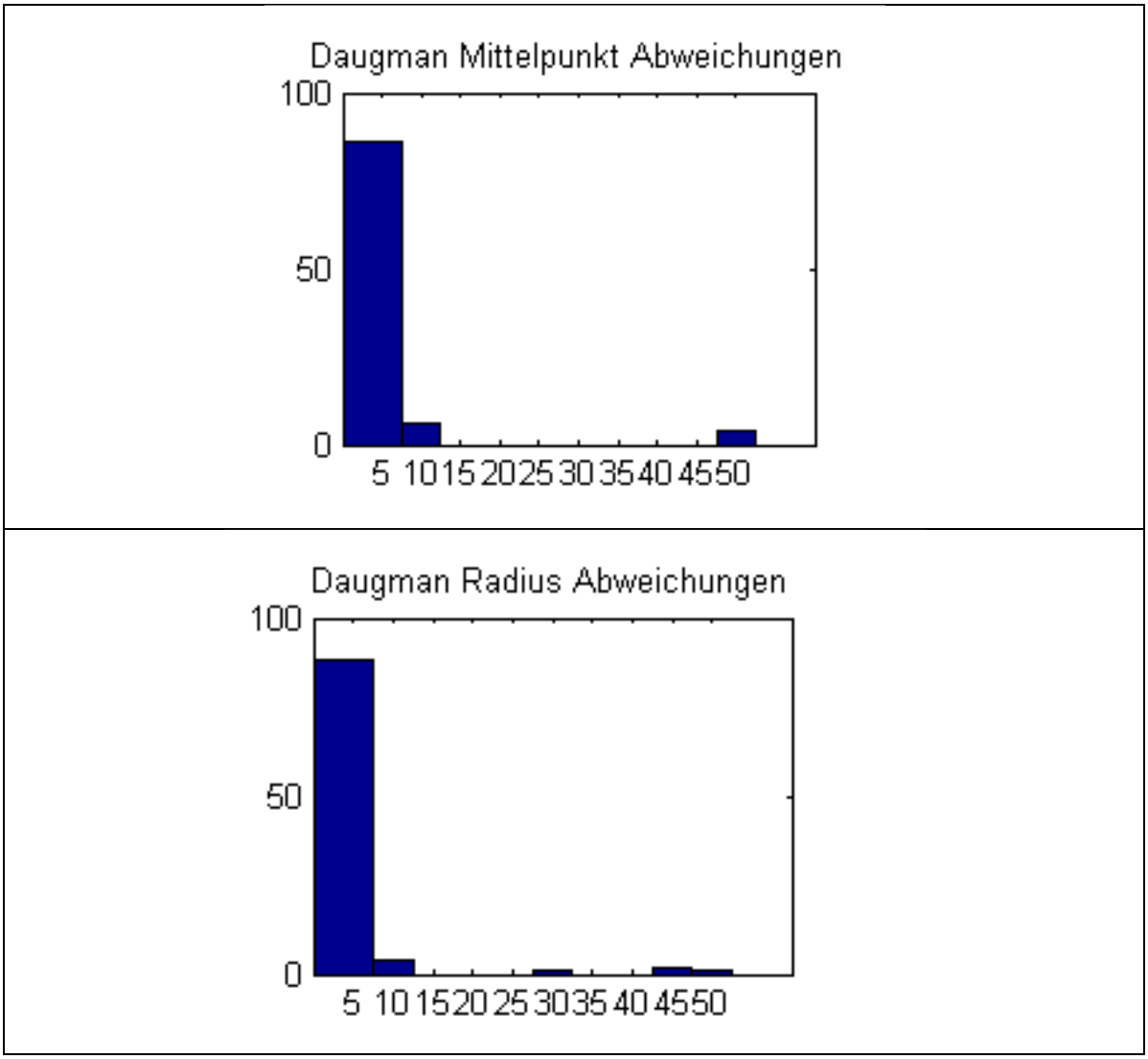
Zusammengefasst entspricht das Pfadintegral in *Gl. 38* der Faltung des Bildes mit den Pfadintegralmasken. Die Integration über den Radius ist mit der Stapelung der Pfadintegralergebnismatrizen in der dreidimensionalen Matrix gleich zu setzten und die Ableitung ist die numerische Differentiation. Die Faltung mit dem Gausskernel in *Gl. 38* ist äquivalent zur Faltung der drei dimensionalen Speichermatrix mit einem ein-dimensionalen Filter in Höhenrichtung.

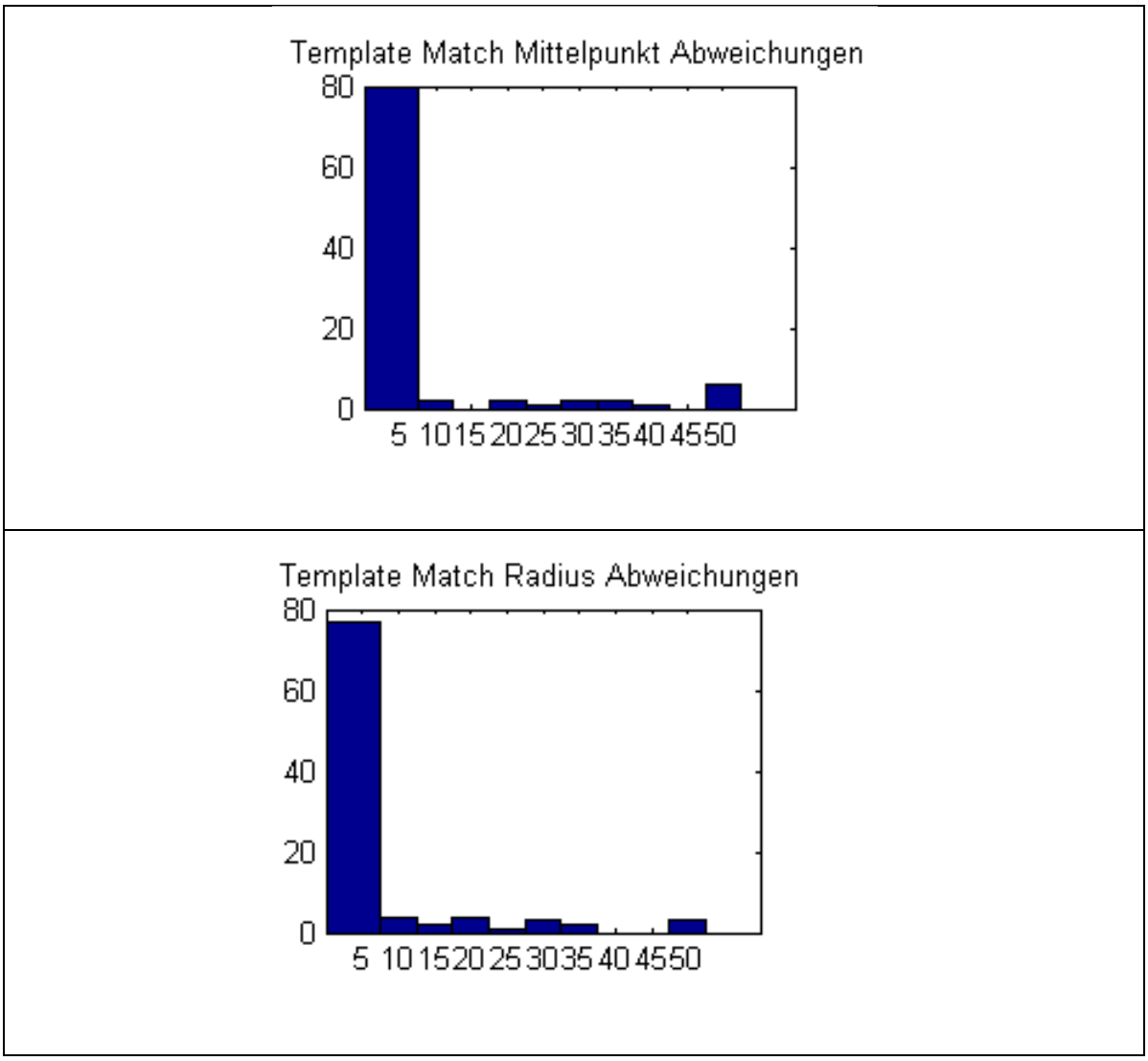
6 Diskussion

Beide Programme sind sehr rechen- und speicheraufwändig. Die Aufwands-bestimmende elementare Rechenoperation ist in beiden Implementierungen das Faltungsprodukt. Ohne Einschränkung des Bereichs (*region of interest*) werden die Programme unter Verwendung der Matlab-Standard-Implementierung der Faltung nicht in absehbarer Zeit fertig. Jeder Versuch diese Berechnung zu beschleunigen ist gescheitert. Ein Ansatz wäre die Faltung in den Frequenzraum zu verlegen. In Matlab kann das innerhalb dieser Arbeit nicht so realisiert werden, dass es schneller ist als die Standard-Funktionen *conv2* und *convn*. Des Weiteren könnten C++ Implementierungen als mex-Dateien dazu geschaltet werden. Dieser Versuch wird aufgegeben, da er den Rahmen der Arbeit überschreiten würde. Eine sehr schnelle Berechnungsmöglichkeit ist mit Nvidia‘s Nsight (http://developer.nvidia.com/nvidia-parallel-nsight) gegeben. Das Projekt integriert die *parallel computing ability* von Grafikprozessoren in Microsoft Visual C++.

6.1 Histogramme für Radius- und Mittelpunktsabweichungen

Die Programme werden mit 109 Bildern getestet. Wie stark die gefundenen Ergebnisse von der von EmcoTest [EMCO-TEST Prüfmaschinen GmbH Brennhoflehen-Kellau 174 A-5431 Kuchl] gelieferten Grundwahrheit abweichen ist nachfolgend aufgetragen (*Histogramm 1* und *Histogramm 2*).





Histogramm 1

In obigem *Histogramm 1* sind alle Abweichungen größer oder gleich 50 bei 50 gruppiert.

Im Vergleich wird ein besseres Abschneiden des Daugman-Integro-Differenzial-Operators deutlich. Man erkennt, dass die Template-Matching-Variante manchmal den Bereich des Kreises zwar gut findet, aber sowohl für den Radius und auch für den Mittelpunkt starke Ungenauigkeiten liefert. – Das bedeutet der Kreis wird zwar ungefähr auf dem Abdruck gefunden, aber es kommt zu einer eher kleineren Abweichung des Mittelpunktes sodass der Kreis etwas verschoben wirkt, sowie zu einer leicht falschen Kreisgröße, derart, dass der Kreis zwar mit einigen Kanten des echten zusammenpasst, an einer Stelle aber ausbricht. Im Gegenzug liefert der Integro-Differential-Operator, wenn er falsche Ergebnisse erzeugt, nur totale Ausreißer, welche völlig neben dem echten Kreis liegen.

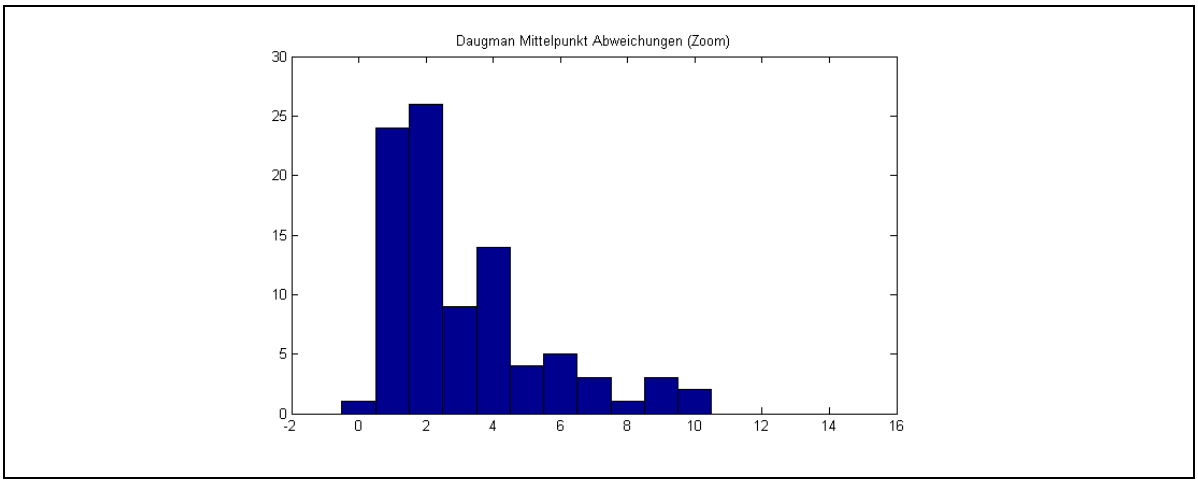
Das Integro-Differential-Operator Programm sollte in weiterer Folge dahingehend erweitert werden, dass nach einem Kreisfund nochmals ein Plausibilitätscheck sowohl durch Bildsegmentierung als auch durch Nachprüfung der Kanten erfolgt. Falls der Kreis

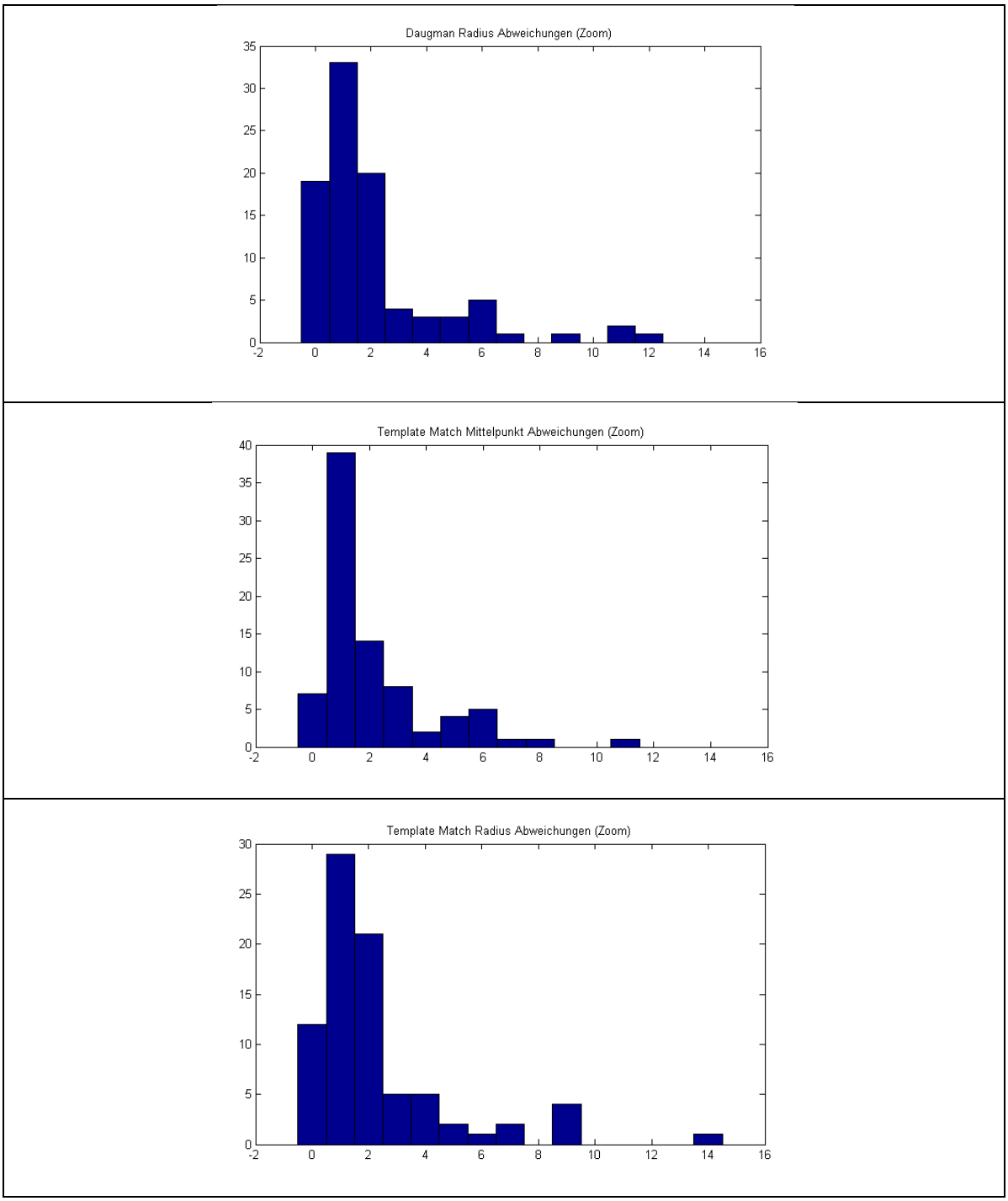
die Prüfung nicht übersteht, wird das entsprechend nächst höhere Maximum herangezogen, welches den Plausibilitätscheck mit besseren Werte übersteht. Hier müsste überlegt werden wie diese Plausibilitätsprüfung auch für exotische Beleuchtungsverhältnisse des Abdrucks umgesetzt werden können, das könnte aber durch eine Kombination von kantenbild und Segmentierungsbild erfolgen. Die Helligkeitsverläufe an der gefundenen Kreiskante könnten auch herangezogen werden um einen Kreisfund zu überprüfen. Es wird geglaubt, dass die Umsetzung solcher Tests fortgeschrittene Ideen aus der Statistik benötigen, die den Kenntnisstand des Autors übersteigen.

Zu Beachten ist die leicht unterschiedliche Skalierung der Ordinatenachse.

Es folgt eine genauere Darstellung von *Histogramm 1* in *Histogramm 2*. Die Bin-Einteilung ist etwas feiner gewählt. – Die interessanten Bereiche werden vergrößert. Die Template-Matching Variante hat auch hier die höheren Fehler. – Besser gesagt sind die Fehler immer größer als die des Integro-Differential-Operators. Es muss angemerkt werden, dass die Mittelpunktsabweichung nach der euklidischen Metrik berechnet wird, also ein Fehler von 10 bei Bildern der Größe 1024 nicht sehr viel ist. Ganz rechts im Radius-Histogramm des Template-Matchings erkennt man die Fehler, die durch Schatten neben dem Abdruck erzeugten Kanten hervorgerufen werden. Das Daugman-Integro-Differential-Operator Programm liefert oft nur ganz leicht zu große Radien, aber keine extremen Ausreisser.

Um beide Programme wirklich zu verbessern wäre es einfach nötig die, Hausnummer, zehn Größten Kreisantworten in der Ergebnismatrix nachträglich statistisch zu vergleichen und nochmals neu zu bewerten, dem Maximum mit dem besten Abschneiden in den geforderten Eigenschaften wird dann der Vorzug gegeben.





Histogramm 2

6.2 Laufzeit und Speicherverbrauchs Diagramme

In nachfolgendem *Diagramm 1* ist die Laufzeit für beide Programme gegen die Größe der *region of interest* aufgetragen. Der Daugman-Integro-Differential-Operator schneidet besser ab. Es könnten glattere Verläufe für die Zeitmessung erzeugt werden, wenn Zeiten für mehrere Messungen gemittelt werden. Die Messung wird auf einem Intel Core2Duo/P8600/Penryn/45nm/2.40GHz (15) durchgeführt.

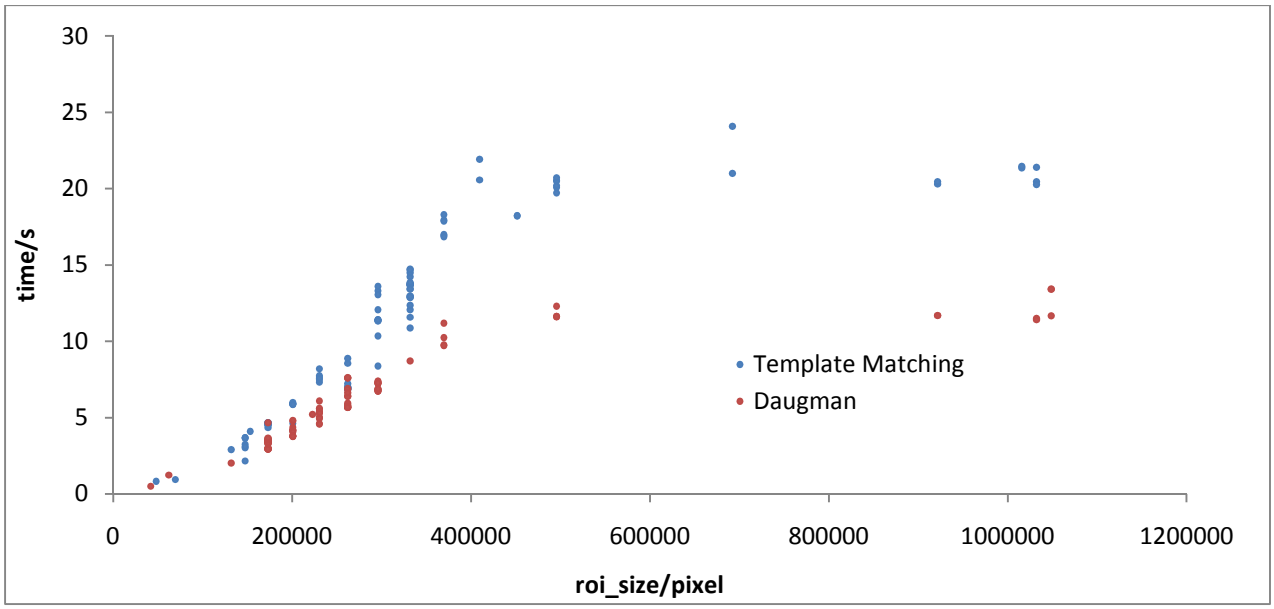


Diagramm 1

Auch beim Speicherverbrauch ist der Daugman-Integro-Differential-Operator besser als die Template-Matching-Variante (*Diagramm 2*).

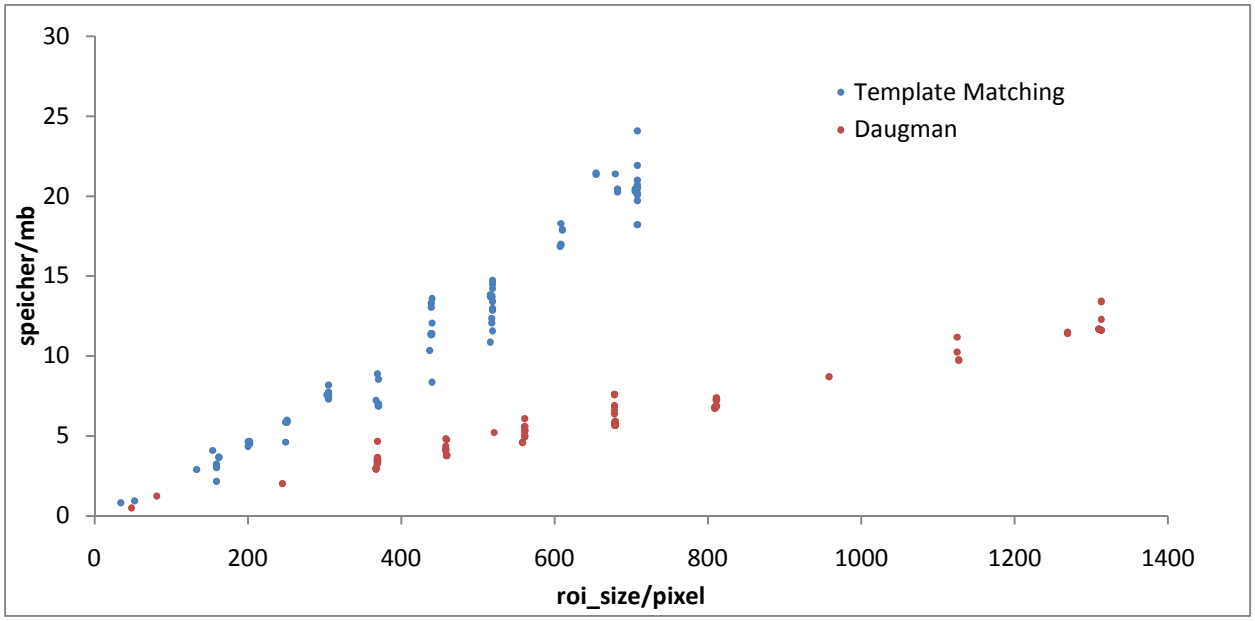


Diagramm 2

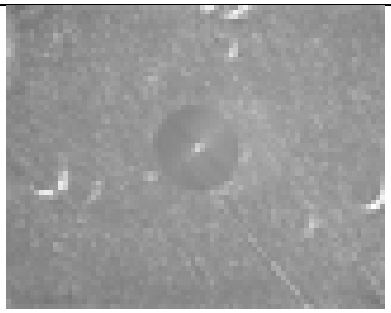
6.3 Analyse der Programm Fehler

6.3.1 Template-Matching Programm

Im Nachfolgenden werden die Fehlerkennungen des Programms aufgelistet. In speziellen Fällen kann es passieren, dass ein Bild mit veränderten Sigmawerten des Canny-Edge-Detectors bzw. Bildhöhen innerhalb der *region of interest*-Suche erkannt wird und dann

nicht als Fehler gewertet werden kann. Im Gegenzug werden dann jedoch andere Bilder nicht mehr richtig erkannt.

Fehlerkennung 1

<table>
<tr><td>erkannte Kreisstelle</td><td></td></tr>
<tr><td>  </td><td>Im Bild rechts ist zu erkennen, dass der Kreis außerhalb des Abdruckes eingezeichnet wird. Im Bild existieren viele helle Stellen, die Kantenerkennung erschweren. Der eigentliche Fehler ist in der region of interest Suche zu finden</td></tr>
<tr><td>region of interset Suchbild nach Filterung und Helligkeitsausgleich</td><td></td></tr>
<tr><td>  </td><td>Das durch den Low-Pass-Filter gelaufene Bild, welches innerhalb der region of interest Suche verwendet wird sieht vielversprechend aus. Das Problem sind die weißen Störungen.</td></tr>
<tr><td>Kantenmaske der region of intersest Suche</td><td></td></tr>
<tr><td>  </td><td>Im durch den Canny-Edge-Detector gefundene Kantenbild ist sofort zu sehen, warum das Template etwas rechts des Kreises besser passt, an dieser Stelle liegen einfach Kanten mit Kreisform vor. Die region of interest wird also bereits falsch erkannt.</td></tr>
<tr><td colspan="2">Was könnte verbessert werden?</td></tr>
<tr><td colspan="2">Tatsächlich kann hier nur eine bessere Vorbearbeitung genauere Ergebnisse bringen. Eine Erhöhung des Canny-Edge-Detector Sigmas in der region of interst-Suche kann zwar für dieses Bild ein genaueres Ergebnis liefern, stört dann aber die Erkennung anderer Bilder. Eine Vergrößerung des Bildes für die region of interst-Suche ist auch nicht möglich, da sonst Bilder des nachfolgenden Typs (Abbildung 53) nicht mehr erkannt werden können.</td></tr>
</table>

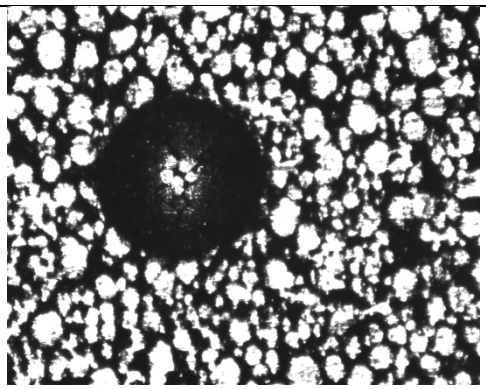
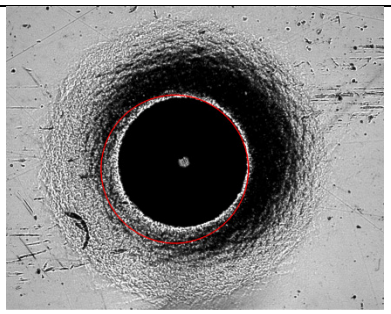
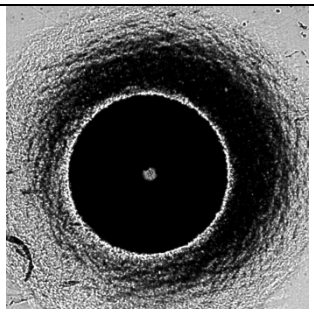
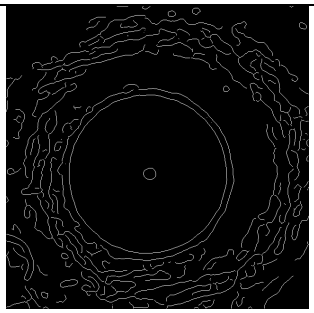


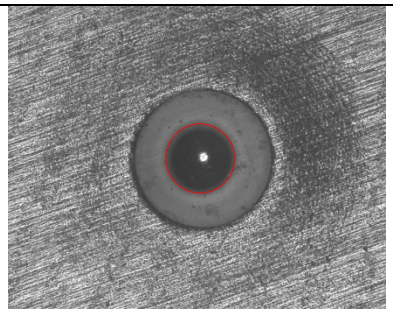
Abbildung 53

Fehlerkennung 2

erkannte Kreisstelle	
	Im linken Bild ist zu sehen, dass der gefundene Kreis zu groß für die Abdruckstelle ist. Dieser Fehler tritt bei richtiger Wahl des Canny-Sigmawertes innerhalb des Template-Matching bei bereits gefundener *region of interest* nicht auf.
gefundene *region of interest*	
	Der interessante Bereich wird eigentlich zunächst gut eingeschränkt, also gibt es innerhalb der *region of interest*-Suche keinen Fehler.
Kantenmaske der *region of interest*	
	Eine Erhöhung des Canny-Edge-Detector Sigmawertes würd hier sicher ein besseres Ergebnis bringen, es kommt dann aber zu einem Genauigkeitsverlust. Dieser Fehler wird mit einem Canny-Sigmawert von 1 erreicht. Es ist ein unglücklicher Zufall, dass das Kreistemplate genau mit den umliegenden Kanten eine bessere Übereinstimmung erzielt.
Was könnte verbessert werden?	

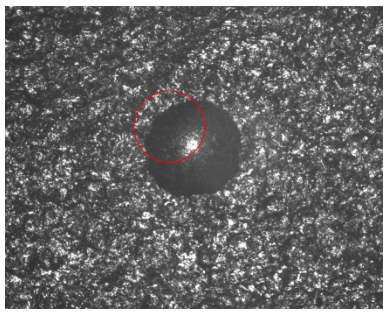
Um eine Verbesserung für genau dieses Bild zu erreichen könnte der Sigma-Wert des Canny-Edge-Detectors erhöht werden, das würde sich dann aber wiederum negativ auf andere Bilder und die Genauigkeit auswirken.

Fehlerkennung 3

erkannte Kreisstelle	
	In diesem Fall liegt der Fehler klar bei der Berücksichtigung konzentrischer Kreise innerhalb der Maximum-Such des Template-Matchings. Der äußere Kreis hat eine geringere Response als der innere.
Man könnte auch konzentrische Kreise berücksichtigen, die eine geringere Antwort als, der innere Kreis liefern. Falls äußere Kreise, die nur 0,9-fach so gut mit einem Kreis zusammen passen wie der Innere wird der Aussenkreis sogar richtig erkannt, dadurch werden in anderen Bildern aber falsche Kreise eingezeichnet.	

6.3.2 Daugman-Integro-Differential-Operator-Programm

Der Daugman-Integro-Differential-Operator ist nicht sehr Fehler-anfällig, es kann jedoch bereits innerhalb der *region of interset* Suche zu Fehlerkennungen kommen.

erkannte Kreisstellen	
 	Wie in den Bildern links zu sehen ist wird der Integro-Differential-Operator durch die ungünstigen hellen Flecken im Bild gestört. Die Gleichverteilung der Werte wird bei der Berechnung des Pfadintegrals nicht berücksichtigt.

Was könnte verbessert werden?
Die Vorverarbeitung müsste verbessert werden, oder es müssten innerhalb der Berechnung der Pfadintegrale gleichverteilte Werte stärker berücksichtigt werden. Es gäbe auch die Möglichkeit den Kantenzug nach dem Kreisfund nochmals auf Plausibilität zu Prüfen und gegebenenfalls ein anderes Kreismaximum zu wählen.

7 Literaturverzeichnis

1. **Matworks.** http://www.mathworks.com. *Matworks.* [Online] Matworks, 1. 1 2011.

2. **http://de.wikipedia.org/wiki/Brute-Force-Methode.** http://de.wikipedia.org/wiki/Brute-Force-Methode. *http://de.wikipedia.org/wiki/Brute-Force-Methode.* [Online] 23. 04 2011.

3. **Daugman, John.** IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 14, NO. 1, JANUARY 2004, How Iris Recognition Works.

4. **http://de.wikipedia.org/wiki/Bresenham-Algorithmus.** http://de.wikipedia.org/wiki/Bresenham-Algorithmus. *http://de.wikipedia.org/wiki/Bresenham-Algorithmus.* [Online] 1. 1 2011.

5. **Canny, John.** A Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

6. **http://www.mathworks.com/help/toolbox/images/ref/adapthisteq.html.** http://www.mathworks.com/help/toolbox/images/ref/adapthisteq.html. *http://www.mathworks.com/help/toolbox/images/ref/adapthisteq.html.* [Online] 24. 3 2011.

7. **Turkey, Cooley and.** An Algorithm for the Machine Calculation of Complex Fourier Series.

8. **A. K. Jain, F. Farrokhnia.** Unsupervised texture segmentation using Gabor filters. *Pattern Recognition, vol. 24, no. 12, pp.1167-1186.* 1991.

9. **Hammouda, Khaled.** Texture Segmentation using Gabor Filters. *Course Project of SD775 at the University of Waterloo, Ontario, Canada.* 1. 5 2003.

10. **Malik, Perona and.** Preattentive texture discrimination with early vision mechanisms. *Opt. Soc. Am. A, Vol. 7, No. 5.* 1. 5 1990.

11. **Seo, Naotoshi.** GaborTextureSegment.pdf. *GaborTextureSegment.pdf.* [Online] 29. 5 2011. [Zitat vom: 29. 5 2011.] http://note.sonots.com/?plugin=attach&refer=SciSoftware%2FGaborTextureSegmentation&openfile=GaborTextureSegment.pdf, erreichbar am 29.5.2011.

12. **D. Clausi, M. Ed Jernigan.** Designing Gabor filters for optimal texture separability. *Pattern Recognition, vol. 33, pp. 1835-1849.* 2000.

13. **Jianguo Zhang, Tieniu Tan, Li Ma.** Invariant texture segmentation via circular gabor filter. *Proceedings of the 16th IAPR International Conference on Pattern Recognition (ICPR), Vol II, pp. 901-904,.* 2002.

14. **Nielsen, Richard Nock and Frank.** Statistical Region Merging. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 26, NO. 11, NOVEMBER 2004 1.*

15. **http://ark.intel.com/Product.aspx?id=35568.** http://ark.intel.com/Product.aspx?id=35568. *http://ark.intel.com/Product.aspx?id=35568.* [Online] 1. 1 2011.

8 Programmtextanhang

```
classdef T45 < hgsetget
    %T45 TEMPLATE MATCHING BASED ROI AND TEMPLATE MATCHING
    %   template matching based roi and template matching



    properties(Constant = true)

        % Innerhalb der Bereichssuche sollte der Startwert des
        % Kreisdurchmessers sehr klein sein. Der minimal zulassige Anteil
        % an der kleinern Bildseite ist 0.3. Um auch die kleinsten Kreise
        % zu finden wird dieser Wert auf 0.3 gesetzt.
        MinimumROISearchStartDiameterFraction = 0.3;
        DefaultROISearchStartDiameterFraction = 0.3;

        % Wenn das ausgeschnittene Bild durchsucht wird darf der kleinste zu
        % suchende Durchmesser einen Anteil von 0.4 nicht unterschreiten,
        % da sonst die Faltungsberechnung zu aufwendig wird.
        MinimumFoundROIStartDiameterFraction = 0.4;
        DefaultFoundROIStartDiameterFraction = 0.4;

        % Der Sigmawert des Canny-Edge Detektors innerhalb der region of
        % interest.
        DefaultROICannySigma = 4;

    end



    properties(Access = private)

        % Es folgt die Definition des Seicherplatzes fuer das Bild. Die
        % Variable wird vom Konstruktor oder der entsprechenden set Methode
        % gesetzt.
        OriginalImage = [];

        % Es folgt die Definition des kleinst moegliche Anteils des
        % Kreisdurchmessers an der kleineren Bildseite fuer die Region of
        % Interest Suche und fuer die anschliessende Suche des Kreises
        % innerhalb der Region of Interest.
        ROISearchStartDiameterFraction = T45.DefaultROISearchStartDiameterFraction;
        FoundROIStartDiameterFraction = T45.DefaultFoundROIStartDiameterFraction;

        WorkingImage = [];

        ROICannySigma = T45.DefaultROICannySigma;

    end



    properties(Access = private, Dependent = true)

        GrayscaleImage = [];
```

```
    end


    methods

        function this = T45(OriginalImage, ROISearchStartDiameterFraction,↙
FoundROIStartDiameterFraction)

            % An den Konstuktor uebergebenes Bild in der Memeber-Variable
            % abspeichern.
            set(this, 'OriginalImage', OriginalImage);

            % Die jeweiligen Startwert in den Membervariablen abspeichern.
            set(this, 'ROISearchStartDiameterFraction',↙
ROISearchStartDiameterFraction);
            set(this, 'FoundROIStartDiameterFraction', FoundROIStartDiameterFraction);

        end



        function this = set.OriginalImage(this, Value)

            % Im nachfolgenden Programmtext wird die Eingabe auf
            % Gueltigkeit geprueft. Es kann sich um ein Graustufenbild oder
            % ein RGB-Bild handeln.
            assert((ndims(Value) == 2) || (ndims(Value) == 3), 'T45::set.OriginalImage:↙
Das Bild (Argument 1) muss ein Graustufenbild oder ein RGB Bild sein.');
            assert(((size(Value, 3) == 1) || (size(Value, 3) == 3)) && (all([size↙
(Value, 1) size(Value, 2)] > [23 23])), 'T45::T45: Das Bild (Argument 1) muss ein↙
Graustufenbild oder ein RGB Bild und mindestens 24x24 in der Groesse sein.');

            % Nach dem die Eingabe ueberprueft worden ist kann das Bild
            % gespeichert werden.
            this.OriginalImage = Value;

        end



        function this = set.ROISearchStartDiameterFraction(this, Value)

            % Die uebergebene Eigenschaft wird nachfolgend auf
            % plausibilitaet geprueft.
            assert(isscalar(Value), 'T45::setROISearchStartDiameterFraction: Die↙
Eigenschaft muss ein Skalar sein.');
            assert(isreal(Value), 'T45::setROISearchStartDiameterFraction: Die↙
Eigenschaft mus reell sein.');
            assert((T45.MinimumROISearchStartDiameterFraction <= Value) && (Value <=↙
1), sprintf('T45::setROISearchStartDiameterFraction: Die Eigenschaft liegt ausserhalb↙
des gueltigen Bereichs von [%d 1].', T45.MinimumROISearchStartDiameterFraction));

            % Falls der Wert korrekt ist wird er in der Eigenschaft der
            % Klasse abgespeichert.
```

```
            this.ROISearchStartDiameterFraction = Value;

        end



        function this = set.FoundROIStartDiameterFraction(this, Value)

            % Die uebergebene Eigenschaft wird nachfolgend auf
            % plausibilitaet geprueft.
            assert(isscalar(Value), 'T45::setFoundROIStartDiameterFraction: Die↙
Eigenschaft muss ein Skalar sein.');
            assert(isreal(Value), 'T45::setFoundROIStartDiameterFraction: Die↙
Eigenschaft mus reell sein.');
            assert((T45.MinimumFoundROIStartDiameterFraction <= Value) && (Value <= 1),↙
sprintf('T45::setFoundROIStartDiameterFraction: Die Eigenschaft liegt ausserhalb des↙
gueltigen Bereichs von [%d 1].', T45.MinimumFoundROIStartDiameterFraction));

            % Falls der Wert korrekt ist wird er in der Eigenschaft der
            % Klasse abgespeichert.
            this.FoundROIStartDiameterFraction = Value;

        end



        function Value = get.GrayscaleImage(this)

            % Falls das Bild mehrer Farbkanaele hat wird ein Graustufenbild
            % erzeugt und zurueckgegeben.
            if(size(this.OriginalImage, 3) == 3)
                Value = rgb2gray(this.OriginalImage);
            else
                Value = this.OriginalImage;
            end

        end

    end



    methods

        function [Result] = Run(this)

            % Im nachfolgenden Codeblock werden die Standardwerte der
            % Rueckgabevarbiable voreingestellt. Bei erfolgreicher
            % Bereichseinschraenkung werden die Werte entspraechend
            % angepassst.
            Result.Success = false;
            Result.OriginalImage = this.OriginalImage;
            Result.CenterRow = 1;
            Result.CenterColumn = 1;
            Result.Diameter = 0;
```

```
            Result.CircleOutlineImage = cat(3, get(this, 'GrayscaleImage'), get(this,↙
'GrayscaleImage'), get(this, 'GrayscaleImage'));
            Result.CircleBoxMask = false(size(this.OriginalImage, 1), size(this.↙
OriginalImage, 2));

            % Es folgen die Definitionen von den benoetigten Bytes fuer
            % Objekte innerhalb dieses Objekts und die Groesse der ROI.
            Result.InternalBytesUsed = 0;
            Result.InternalROISize = [0 0];

            this.WorkingImage = get(this, 'GrayscaleImage');

            % Nachfolgend wird die Beleuchtung des Bildes angepasst.
            %Deillum = DEILLUM(this.WorkingImage, size(this.WorkingImage, 1));
            %DeilluminatedImage = Deillum.Run();
            %this.WorkingImage = DeilluminatedImage;

            % Der Speicherverbrauch der Degrind Klasse wird
            % ermittelt um ihn in der Rueckgabestruktur zu speichern.
            %DeillumToStruct =  builtin('struct', Deillum); %#ok<NASGU>
            %DeillumBytesUsed = whos('DeillumToStruct', 'bytes');
            %DeillumBytesUsed = DeillumBytesUsed.bytes;
            %clear DeillumToStruct;
            %clear Deillum;

            % Das Bild wird vor der region of interest suche von einer
            % unguenstigen, starken Schliffspuren befreit. Das Verfahren
            % wird fuer die Bereichsuche durchgefuhert.
            %Degrind = DEGRIND(this.WorkingImage, size(this.WorkingImage, 1));
            %DegrindedImage = Degrind.Run();
            %this.WorkingImage = DegrindedImage;

            % Der Speicherverbrauch der Degrind Klasse wird
            % ermittelt um ihn in der Rueckgabestruktur zu speichern.
            %DegrindToStruct = builtin('struct', Degrind); %#ok<NASGU>
            %DegrindBytesUsed = whos('DegrindToStruct', 'bytes');
            %DegrindBytesUsed = DegrindBytesUsed.bytes;
            %clear Degrind;
            %clear DegrindToStruct;

            % Es wird die Region Interest Suche mit template matching
            % gestartet. Das Bild wird dabei signifikant verkleinert.
            Roi = ROI(this.WorkingImage, this.ROISearchStartDiameterFraction);
            ROISearchResult = Roi.Run();

            % Der Speicherverbrauch der Region of Interest Klasse wird
            % ermittelt um ihn in der Rueckgabestruktur zu speichern.
            RoiToStruct = builtin('struct', Roi); %#ok<NASGU>
            RoiBytesUsed = whos('RoiToStruct', 'bytes');
            RoiBytesUsed = RoiBytesUsed.bytes;
            clear Roi;
            clear RoiToStruct;

            % Falls die Region of Interest Suche fehlschlaegt, dann ist der
            % Durchmesser 0 und das Programm muss agbebrochen werden.
```

```
            if(~ROISearchResult.Success)
                fprintf('T45::Run: unbekannter Fehler bei der Bereichseinschraenkung↙
...\n');
                return;
            else

                fprintf('T45::Run: region of interest gefunden, Kreisssuche starten ...↙
\n');

                % Falls die Bereichseinschraenkung erfolgreich verlaufen
                % ist wird das Template Matching nochmals auf den
                % eingeschraenkten Bereich angewendet, dabei kann der
                % Region of Interest Bereich eine Groesse von 640x640
                % ueberschrieten, ist das der Fall muss auf diese Groesse
                % eingschraenkt werden.
                if(any(size(ROISearchResult.ROIImage) > [640 640]))

                    fprintf('T45::Run: region of interest exceeds maximum size allowed,↙
scaling down ...\n');

                    [~, MaximumSideIndex] = max(size(ROISearchResult.ROIImage));
                    ScaleDownVector = [NaN NaN];
                    ScaleDownVector(1, MaximumSideIndex) = 640;
                    ScaleDownROI = imresize(ROISearchResult.ROIImage, ScaleDownVector,↙
'bicubic');
                    Tm = TM(ScaleDownROI, size(ScaleDownROI, 1), this.↙
FoundROIStartDiameterFraction, 'feature', this.ROICannySigma);
                    FoundROIResult = Tm.Run();
                    FoundROIResult.CircleOutlineImage = imresize(FoundROIResult.↙
CircleOutlineImage, size(ROISearchResult.ROIImage), 'bicubic');
                    FoundROIResult.CircleBoxMask = imresize(FoundROIResult.↙
CircleBoxMask, size(ROISearchResult.ROIImage), 'bicubic');
                    FoundROIResult.CenterRow = round(FoundROIResult.CenterRow * size↙
(ROISearchResult.ROIImage, 1)/size(ScaleDownROI, 1));
                    FoundROIResult.CenterColumn = round(FoundROIResult.CenterColumn *↙
size(ROISearchResult.ROIImage, 2)/size(ScaleDownROI, 2));
                    FoundROIResult.Diameter = round(FoundROIResult.Diameter * size↙
(ROISearchResult.ROIImage, 2)/size(ScaleDownROI, 2));

                else
                    ROIImage = ROISearchResult.ROIImage;
                    Tm = TM(ROIImage, size(ROISearchResult.ROIImage, 1), this.↙
FoundROIStartDiameterFraction, 'feature', this.ROICannySigma);
                    FoundROIResult = Tm.Run();
                end

                % Der Speicherverbrauch der Daugman Klasse wird
                % ermittelt um ihn in der Rueckgabestruktur zu speicheren.
                TmToStruct = builtin('struct', Tm); %#ok<NASGU>
                TmBytesUsed = whos('TmToStruct', 'bytes');
                TmBytesUsed = TmBytesUsed.bytes;
                clear Dman;
                clear DmanToStruct;

                % Das Endergebnis berechnen und in der Rueckgabevariable
```

```
                % abspeichern. Der Offset der Region of Interest muss dazu
                % addiert werden.
                Result.Success = true;
                Result.CenterRow = FoundROIResult.CenterRow + min(ROISearchResult.↙
ROIRows) - 1;
                Result.CenterColumn = FoundROIResult.CenterColumn + min↙
(ROISearchResult.ROIColumns) - 1;
                Result.Diameter = FoundROIResult.Diameter;
                Result.CircleBoxMask(ROISearchResult.ROIRows, ROISearchResult.↙
ROIColumns) = FoundROIResult.CircleBoxMask;

                % Den Kreis nochmals in das Bild einzeichnen und in der
                % Rueckgabe abspeichern.
                Red = get(this, 'GrayscaleImage');
                Green = get(this, 'GrayscaleImage');
                Blue = get(this, 'GrayscaleImage');
                Red(Result.CircleBoxMask) = 255;
                Green(Result.CircleBoxMask) = 0;
                Blue(Result.CircleBoxMask) = 0;
                Result.CircleOutlineImage = cat(3, Red, Green, Blue);

                %Result.InternalBytesUsed = DeillumBytesUsed + DegrindBytesUsed +↙
RoiBytesUsed + ROISearchResult.InternalBytesUsed;
                Result.InternalBytesUsed = RoiBytesUsed + ROISearchResult.↙
InternalBytesUsed + TmBytesUsed;
                Result.InternalROISize = [numel(ROISearchResult.ROIRows), numel↙
(ROISearchResult.ROIColumns)];

            end

        end

    end

end
```

```
classdef TM < BHAM
    %TM TEMPLATE MATCHING CORE
    %   template matching core

    properties(Constant = true)

        % Nachfolgend werden die Standardbeschraenkungen fuer Bildhoehe und
        % Bildbreite definiert. Diese Werte sind obere Grenzen. Der
        % Bruchteil des kleinsten Radius an der kleineren Seite ist nach
        % unten beschraenkt, Werte groesser als 1 ergeben auch keinen Sinn.
        % Die MinimumStartDiameterFraction muss mit den kleinsten
        % Bildgroessen von 24 konsistent sein und sollte daher nicht
        % kleiner als 0.3 werden.
        MaximumHeight = 640;
        MaximumWidth = 640;
        MinimumStartDiameterFraction = 0.3;
        DefaultMethod = 'feature'; % Mercedes Verkehrszeichen Methode, muss gut sein!
        DefaultPadding = 0;
        DefaultThickness = 2;
        DefaultTemplateUseBackground = false;
        DefaultFeatureCannySigma = 1;

    end



    properties(Access = protected)

        % Es folgt der Durchmesserstartwert, als Anteil der kleineren
        % Seite. Die Variable wird vom Konstruktor gesetzt.
        StartDiameterFraction = 0;
        Method = TM.DefaultMethod;
        Padding = TM.DefaultPadding;
        Thickness = TM.DefaultThickness;
        TemplateUseBackground = TM.DefaultTemplateUseBackground;

        % Die nachfolgenden Variablen werden von der Funktion
        % TM::LoadTemplates gesetzt.
        StartDiameter = 0;
        EndDiameter = 0;
        Templates = 0;
        MatchResultStack = [];

        % Nachfolgend werden die Speicherplaetze fuer die Bilder, die
        % gematcht werden sollen definiert. Das DistanceImage wird nur im
        % Falle der feature Methode benoetigt.
        EdgeMask = [];
        WorkingImage = [];

        % Es folgt die standard Definiton des Sigmawerts fuer die Canny
        % Kantenerkennung, innerhalb der feature Methode.
        FeatureCannySigma = TM.DefaultFeatureCannySigma;

    end
```

```
    methods

        % Konstruktor der TM Klasse. Dem Konstruktor muss das zu
        % bearbeitende Bild und die Verarbeitungsgroesse des Bildes
        % uebergeben werden. Die Groesse darf die MaximumHeight nicht
        % ueberschrieten.
        function this = TM(Image, Height, StartDiameterFraction, varargin)

            % Das Bild muss ein Graustufenbild sein. Ein Bild mit drei
            % Farbkannaelen wird nicht akzeptiert. Das Bild muss mindestens
            % 24 mal 24 Groesse haben.
            assert(ndims(Image) == 2, 'TM::TM: Das Bild (Argument 1) muss ein↙
Graustufenbild sein.');
            assert(all(size(Image) > [23 23]), 'TM::TM: Das Bild (Argument 1) muss↙
mindestens eine Groesse von 24x24 haben.');

            % Die angegebene Hoehe darf die maximale Hohe nicht
            % ueberschreiten und darf nicht kleiner als 24 sein.
            assert((round(Height) == Height) && (Height < (TM.MaximumHeight + 1)) &&↙
(Height > 23), sprintf('TM::TM: Die Bildhoehe muss ganzzahlig, mindestens 24 und↙
hoechstens %d sein.', TM.MaximumHeight));

            % Das Bild wird verkleinert und es wird geprueft, ob die maximal
            % zulaessige Breite nicht ueberschritten wird.
            Image = imresize(Image, [Height NaN], 'bicubic');
            assert((size(Image, 2) < (TM.MaximumWidth + 1)) && (size(Image, 2) > 23),↙
sprintf('TM::TM: Die Bildbreite darf %d nicht ueberschreiten und nicht kleiner als 24↙
sein. Sie ist %d.', TM.MaximumWidth, size(Image, 2)));

            % Der Startwert des Durchmessers wird als Antiel der kleineren
            % Seite angegeben. Er darf nicht grosser als 1 und kleiner als
            % der minimale Wert sein.
            assert((StartDiameterFraction <= 1) && (StartDiameterFraction >= TM.↙
MinimumStartDiameterFraction), sprintf('TM::TM: Der Anteil des Anfangsdurchmessers an↙
der kleineren Seite muss zwischen 1 und %d liegen.', TM.MinimumStartDiameterFraction));

            % Wenn alles in Ordnung ist, wird das Bild abgespeichert.
            this.WorkingImage = Image;
            this.StartDiameterFraction = StartDiameterFraction;

            % Nun wird die optionale Eingabe geprueft. Falls keine
            % optionalen Parameter uebergeben werden, dann ist alles
            % standard.
            if(size(varargin, 2) > 0 && size(varargin, 2) < 3)
                assert(strcmp(varargin{1}, 'feature') || strcmp(varargin{1},↙
'template'), 'TM::TM: Method (optionales Argument 1) muss entweder feature oder↙
template sein.');
                this.Method = varargin{1};
                % Wenn die Methode feature ist, dann kann der Sigma Wert
                % gesetzt werden.
                if(strcmp(this.Method, 'feature') && size(varargin, 2) == 2)
                    assert(isnumeric(varargin{2}) && (varargin{2} >= 0.1) && (varargin↙
{2} <= 20), 'TM::TM: Canny Sigma ist keine Zahl zwischen 0.1 und 20.')
```

```
                this.FeatureCannySigma = varargin{2};
            else
                assert(false, 'TM::TM: Falsche optionale Parameteranzahl fuer↙
Methode feature.');
            end
        end

        % Das Kantenbild wird berechnet.
        this.BuildEdgeMask();

    end

end



methods(Access = public)

    function [Result] = Run(this)

        % Zuerst wird versucht die Templates aus einer Datei zu laden.
        % Falls sie vorher noch nicht erzeugt und gespeichert wurden,
        % werden sie neu erstellt und in eine, File abgelegt.
        Loaded = this.LoadTemplates();
        if(~Loaded)
            this.ProduceTemplates();
            this.SaveTemplates();
        end

        % Nachfolgend werden alle erzeugten Templates mit dem
        % jeweiligen Bild gematched.
        this.RunMatching();

        % Nach der Berechnung der Faltungen werden die Spalten und
        % Zeilen Indexe des globalen Maximums gesucht.
        Best = this.FindMaximumIndexes();

        % Die Ergebnisse ausgeben. Ein Durchmesser von Null deutet
        % einen nicht sinnvollen Kreisfund an. Ansonsten wird der
        % Umfang des Kreises eingezeichnet. Dazu wird der Bresenham
        % Algorithmus verwendet. Der Kreisfund kann auch dann sinnlos
        % sein wenn der Durchmesser nicht Null ist.
        Result.WorkingImage = this.WorkingImage;
        Result.CenterRow = Best.Row;
        Result.CenterColumn = Best.Column;
        if(Best.Diameter ~= 0)

            % Der beste Radius wird in der Rueckgabe Struktur
            % gespeichert. Dieser Durchmesser ist hier sicher >= 3.
            Result.Diameter = Best.Diameter;

            % Die Aussenlinie des Kreises muss noch in das Bild
            % eingezeichnet werden. Dazu wird der Bresenham-Algorithmus
            % verendet. Das Aussenlinienbild ist ein RGB Bild. Das in
            % der Ergebnisstruktur gespeicherte CircleOutlineImage ist
```

```
                % nicht unbedingt geignet um Ergbnisse im echten Bild
                % darzustellen. Je nachdem wie vorverarbeitet wurde oder ob
                % die Methode feature oder template gewaehlt wurde, sollte
                % die CircleBoxMask dazu verwendet werden den Kreis in das
                % echte zu verarbeitende Bild zu zeichnen.
                CircleBoxMask = TM.BresenhamCircleBoxMask(Best.Diameter, 2, [Best.Row,↙
Best.Column], size(this.WorkingImage));
                Red = mat2gray(this.WorkingImage);
                Green = mat2gray(this.WorkingImage);
                Blue = mat2gray(this.WorkingImage);
                Red(CircleBoxMask) = 255;
                Green(CircleBoxMask) = 0;
                Blue(CircleBoxMask) = 0;
                Result.CircleBoxMask = CircleBoxMask;
                Result.CircleOutlineImage = cat(3, Red, Green, Blue);

            else
                Result.Diameter = 0;
                Result.CircleBoxMask = false(this.WorkingImage);
                Result.CircleOutlineImage = cat(3, this.WorkingImage, this.↙
WorkingImage, this.WorkingImage);
            end


        end

    end



    methods(Access = private)

        % Diese Funktion wird als erstes gestartet. Es wird die Anzahl der
        % benoetigten Kreistemplates ermittelt. Falls eine Datei mit diesen
        % Templates im Arbeitsverzeichnis bereits exisitiert, wird sie
        % geladen und die Templates muessen nicht neu erzeugt werden.
        function [Loaded] = LoadTemplates(this)

            % Nachfolgend wird die kleinere der Bildseiten ermittelt und
            % der Start und Endwert des Abdruckdurchmessers berechnet.
            SmallerSide = min(size(this.WorkingImage));
            this.StartDiameter = round(SmallerSide * this.StartDiameterFraction);
            this.StartDiameter = this.StartDiameter + double(~bitget(this.↙
StartDiameter, 1));
            this.StartDiameter = this.StartDiameter;
            this.EndDiameter = SmallerSide;
            this.EndDiameter = this.EndDiameter - double(~bitget(this.EndDiameter, 1));

            % Es wird nun eine Array mit der benoetigten Groesse erstellt.
            % Der Array hat eine Zeile und die geforderte Anzahl an
            % Spalten. Die Groesse kann immer mit size abgefragt werden.
            this.Templates = cell(1, ((this.EndDiameter - this.StartDiameter)/2) + 1);

            % Nachfolgend wird der Dateiname der gesuchten Maskendatei
            % zusammengestellt und danach die Datei geladen.
```

```
            FileName = sprintf('tm.%d.%d.%s.mat', this.StartDiameter, this.EndDiameter,↙
this.Method);
            if(exist(FileName, 'file'))
                fprintf('TM::LoadTemplates: mask file exists, loading templates from↙
workspace ...\n');
                load(FileName, 'Masks');
                this.Templates = Masks;
                clear Masks;
                Loaded = true;
            else
                fprintf('TM::LoadTemplates: mask file does not exists, cannot load↙
something ...\n');
                Loaded = false;
            end

        end



        % Diese Methode erzeugt die noetigen Templates, je nach
        % ausgewaelter template matching Art. Ein vorheriges Laufen von
        % TM::LoadTemplates ist erforderlich.
        function [] = ProduceTemplates(this)

            % Je nach Methode werden die Templates anders aufgebaut.
            switch this.Method

                % Falls die feature Method gewaehlt wurd ist die
                % Erzeugung der Templates einfacher als innerhalb der
                % template Methode
                case 'feature'

                    % Alle noetigen Durchmesser werden in einem Vektor
                    % zusammengetragen und durchlaufen.
                    Diameters = this.StartDiameter:2:this.EndDiameter;
                    for i = 1:1:size(this.Templates, 2)

                        fprintf('TM::ProduceTemplates: producing template %d/%d ...\n',↙
i, size(this.Templates, 2));

                        % Zuerst muss geprueft werden, ob der
                        % Kreisdurchmesser mit dem angegbenen Rahmen noch
                        % in die kleiner Bildseite passt. Wenn das nicht
                        % der Fall ist wird der Rahmen reduziert.
                        if((2 * this.Padding + Diameters(1, i)) < (min(size(this.↙
WorkingImage)) + 1))
                            ValidPadding = this.Padding;
                        else
                            ValidPadding = floor((min(size(this.WorkingImage)) -↙
Diameters(1, i))/2);
                        end

                        % Die Kreismaske wird mit dem Bresenham Algorithmus
                        % berechnet. Die Methode wird vererbt.
                        CircleMask = TM.BresenhamCircleMask(Diameters(1, i), this.↙
```

```
Thickness, ValidPadding);

                        % Die Templates werden abgespeichert. Welche
                        % Bereiche als Wildcards ausgeblendet werden wird
                        % in einer Maske festgehalten.
                        this.Templates{1, i}.WildcardsMask = ~ CircleMask;
                        this.Templates{1, i}.Template = double(CircleMask);
                        this.Templates{1, i}.Template(this.Templates{1, i}.↙
WildcardsMask) = 0;
                        this.Templates{1, i}.TemplateDiameter = Diameters(1, i) + 2 *↙
ValidPadding;
                        this.Templates{1, i}.CircleDiameter = Diameters(1, i);

                    end

                case 'template'

                    % This is a stub for implementation of real template
                    % matching.

            end

        end



        % Diese Funktion speichert die erzeugten Templates in einer Datei
        % ab. Ein vorheriges Laufen von TM::ProduceTemplates ist noetig.
        function [] = SaveTemplates(this)

            % Nun wird der Dateiname zusammengestellt. Falls die Datei im
            % Arbeitsverzeichnis bereits existiert, muss sie nicht extra
            % neu gespeichert werden.
            FileName = sprintf('tm.%d.%d.%s.mat', this.StartDiameter, this.EndDiameter,↙
this.Method);
            if(exist(FileName, 'file'))
                fprintf('TM::SaveTemplates: mask file exists, no need to save masks ...↙
\n');
            else
                fprintf('TM::SaveTemplates: writing masks to file ...\n');
                Masks = this.Templates; %#ok<NASGU>
                save(FileName, 'Masks');
                clear Masks;
            end

        end



        % Durchlauft alle Templates und ruft fuer jedes Template die
        % MatchTemplate Methode auf. Die Ergbnisse werden in einer drei
        % dimensionalen Matrix abgespeichert.
        function [] = RunMatching(this)

            this.MatchResultStack = zeros([size(this.WorkingImage, 1), size(this.↙
```

```
WorkingImage, 2), size(this.Templates, 2)]);
            for i = 1:1:size(this.Templates, 2)

                fprintf('TM::RunMatching: matching template %d/%d ...\n', i, size(this.↙
Templates, 2));

                this.MatchTemplate(i, 'cocorrelation');

                %Box = zeros(this.Templates{i}.TemplateDiameter, size(this.↙
WorkingImage, 2));
                %Box(1:this.Templates{i}.TemplateDiameter, 1:this.Templates{i}.↙
TemplateDiameter) = this.Templates{i}.Template;
                %CombinedImage = cat(1, mat2gray(this.EdgeMask), mat2gray(Box));
                %ColorCombinedImage = cat(3, CombinedImage, CombinedImage,↙
CombinedImage);
                %[x, y] = meshgrid(1:1:64, 1:1:64);
                %z = this.MatchResultStack(:, :, i);
                %z = imresize(z, [64 64], 'nearest');
                %gf = fspecial('gaussian', [64, 64], 1);
                %z = conv2(z, gf, 'same');
                %surf(x, y, z); colormap(jet); colorbar;
                %surf(x, y, z); colormap(jet); colorbar;
                %uiwait;
                %imshow(ColorCombinedImage);
                %uiwait;
                %subplot(1, 2, 1), surf(x, y, z); colormap(jet); colorbar;
                %subplot(1, 2, 2), imshow(ColorCombinedImage);
                %uiwait;

            end

        end



        % Fuert den template matching arlgorithmus je nach andgebener
        % Methode aus und speichert das Ergbnis in der Klasseneigenschaft
        % ab. Der Index des jeweilgien Templates muss uebergeben werden.
        function [] = MatchTemplate(this, i, MatchMethod)

            switch MatchMethod

                % Die Kokorrelationsmethode wird zum Matchen der Templates
                % verwendet. Die gesamte Berechnung laesst sich auf
                % Faltungsprodukt-Berechnungen reduzieren.
                case 'cocorrelation'

                    % Zunaechst wird die Zaehler-Matrix berechnet. Fuer
                    % alle moeglichen Stellen an denen das Template
                    % probiert werden kann ergibt sich dann ein Nenner. Das
                    % Ergebnis ist also eine Matrix.
                    NumeratorMatrix = conv2(this.EdgeMask, this.Templates{1, i}.↙
Template, 'valid');
                    NumeratorMatrix = NumeratorMatrix.^2;
```

```
                    % Das Template wird nun quadriert und die Summe aller
                    % Elemente berechnet. Im Falle der feature Methode,
                    % ist das Quadrieren eher uebrfluessig, da 1 quadriert
                    % 1 ergibt. Fuer die template Methode ist das
                    % Quadrieren aber unbedingt noetig. Nachtraeglich wird
                    % die Summe der Elemente des quadrierten Templates
                    % berechnet. Um den Nenner des Bruchs zu berechnen muss
                    % ein neuere Faltungskernel berechnet werden. Innerhalb
                    % dieses Faltungskernels muessen die Wildcards
                    % ausgeblendet werden, sprich das Negativ wird gesetzt.
                    % Sollten alle Wildcards 0 sein, kann NaN im
                    % Endergebnis auftreten.
                    SquaredTemplate = this.Templates{1, i}.Template .* this.Templates↙
{1, i}.Template;
                    SumSquaredTemplate = sum(sum(SquaredTemplate));
                    DenominatorConvolutionKernel = zeros(this.Templates{1, i}.↙
TemplateDiameter);
                    DenominatorConvolutionKernel(:, :) = SumSquaredTemplate;
                    DenominatorConvolutionKernel(this.Templates{1, i}.WildcardsMask) =↙
0;

                    % Das Arbeits-Bild wird nun quadriert und der Zaehler
                    % kann durch Faltung berechnet werden.
                    SquaredWorkingImage = this.EdgeMask .* this.EdgeMask;
                    DenominatorMatrix = conv2(SquaredWorkingImage,↙
DenominatorConvolutionKernel, 'valid');
                    %DenominatorMatrix = sqrt(DenominatorMatrix);

                    % Da das Kantenbild auch Wildcards haben kann, müssen
                    % alle Pixel, die in der DenominatorMatrix 0 sind
                    % entsprechend behandelt werden. Sie muessen im
                    % Zaehler auch 0 sein und werden, daher im Nenner auf 1
                    % gesetzt.
                    DenominatorMatrix(DenominatorMatrix == 0) = 1;

                    % Das Ergebnis des Matchingerfahrens is eine Matrix, in
                    % der das globale Maximum die am besten passende Stelle
                    % andeutet. Die Ergbnismatrix wird auf Bildgroesse
                    % angepasst. Alle nicht validen Bereiche sind minimal.
                    ValidResultMatrix = NumeratorMatrix./DenominatorMatrix;
                    Offset = (this.Templates{1, i}.TemplateDiameter - 1)/2;
                    this.MatchResultStack(Offset:(Offset + size(ValidResultMatrix, 1) -↙
1), Offset:(Offset + size(ValidResultMatrix, 2) - 1), i) = ValidResultMatrix;

                    fprintf('TM::MatchTemplate: template maximum is %f ...\n', max(max↙
(this.MatchResultStack(:, :, i))));
            end

        end



        function [Best] = FindMaximumIndexes(this)

            % Der Rueckgabewert ist eine Struktur mit den Daten der besten
```

```
            % Maximalstelle ueber alle moeglichen Mittelpunkte. Wenn das
            % WorkingImage tatsaechlich nur aus Nullen besteht wird das
            % Maximum als Null zurueckgegeben. Der Mittelpunkt ist dann im
            % linken oberen Eck und der Durchmesser des Kreises ist Null.
            Best.Maximum = 0;
            Best.Diameter = 0;
            Best.Row = 1;
            Best.Column = 1;

            % Nachfolgend werden die globalen Maxima der dritten
            % Dimension ermittelt, also die Maxima die in fuer verschiedene
            % Template Durchmesser und diamt Kreisdurchmesser auftreten.
            [ThirdMaxima, ThirdIndexes] = max(this.MatchResultStack, [], 3);
            [ColumnMaxima, RowIndexes] = max(ThirdMaxima, [], 1);
            [Maximum, ColumnIndex] = max(ColumnMaxima);

            % Es ist moeglich, dass konzentrische Kreise auftreten, in
            % diesem Fall muss am selben Mittelpunkt nach einer hohen
            % Antwort gesucht werden.
            OtherMaxima = this.MatchResultStack(RowIndexes(ColumnIndex), ColumnIndex,↙
ThirdIndexes(RowIndexes(ColumnIndex), ColumnIndex):end);
            OtherMaximaFraction = OtherMaxima./Maximum;
            OtherMaximaIndexes = (OtherMaximaFraction >= 1);
            if(any(OtherMaximaIndexes))
                fprintf('TM::FindMaximumIndexes: %d other maxima are present ...\n',↙
numel(OtherMaximaIndexes))
                Maximum = max(OtherMaxima(OtherMaximaIndexes));
            end

            % Das globale Maximum aller Template grossen und aller
            % moeglichen Stellen wird als die Mittelpunktsstelle gefunden.
            % Der Durchmesser ist der Kreisdurchmesser des Templates mit
            % dem Index identisch der dritten Koordinate des lokalen
            % Maximums.
            Best.Maximum = Maximum;
            Row = RowIndexes(ColumnIndex);
            Column = ColumnIndex;
            Best.Row = Row;
            Best.Column = Column;
            Best.Diameter = this.Templates{ThirdIndexes(RowIndexes(ColumnIndex),↙
ColumnIndex)}.CircleDiameter;

            % Die besten Werte werden auf der Standardausgabe angezeigt.
            fprintf('TM::FindMaximumIndexes: best diameter is %d at row %d and column %↙
d ...\n',  Best.Diameter, Best.Row, Best.Column);

        end

    end



    methods(Access = protected)

        function [] = BuildEdgeMask(this)
```

```
            this.EdgeMask = edge(this.WorkingImage, 'canny', [], this.↙
FeatureCannySigma);
            %DistanceTransform = bwdist(this.EdgeMask) + 1;
            %DropByDistance = 1./DistanceTransform.^1;
            %DropByDistance = 128 .* DropByDistance;
            %DropByDistance(DropByDistance < 16) = 0;
            %this.EdgeMask = double(this.EdgeMask) .* 256 + double(~this.EdgeMask) .*↙
DropByDistance;
            this.EdgeMask = double(this.EdgeMask);

        end

    end

end
```

```
  classdef ROI < TM
    %ROI TEMPLATE MATCHING BASED REGION OF INTEREST SEARCH
    %   template matching based region of interest search

    properties(Constant = true)

        % Die Regionssuche benoetigt nur eine geringe Groesse. Der Kreis
        % sollte grob detektiert werden und alle Textur oder Schliffeffekte
        % sollten durch die Verkleinerung und die damit einhergehende
        % Interpolation verschwinden.
        SearchHeight = 64;

        % Die Region of Interest ist ein eckiger Bereich, gefunden wird
        % aber ein Kreis. Diese Variable legt fest, um wieviele Zeilen bzw.
        % Spalten die Region of Interest in jede Richtung innerhalb der
        % kleinen Maske verkleinert bzw. vergroessert wird.
        ExtendROI = 2;

        % Es folgt die Definition der Standardabweichung des Gaussfilters
        % fuer den Canny-Algorithmus. Es wird immer Kanten-Matching
        % verwendet.
        CannySigma = 3;

    end



    properties(Access = private)

        % das originale Bild wird in seiner oringalen Groesse gespeichert.
        % Der gefundene Bereich, als die Region of Interest muss diesem
        % Bild, also dessen Groesse angepasst werden.
        OriginalImage = 0;

    end



    methods

        function this = ROI(Image, StartDiameterFraction)

            % Es wird der Konstruktor der uebergeordneten Klasse
            % aufgerufen. Die Parameter werden entsprechend gewaehlt.
            this = this@TM(Image, ROI.SearchHeight, StartDiameterFraction, 'feature',↙
ROI.CannySigma);

            % An dieser Stelle liegt Image sicher als Graustufenbild vor.
            % Der Konstruktor von der Ueberklasse versichert das.
            this.OriginalImage = Image;

        end

    end
```

```
    methods(Access = public)

        function [Result] = Run(this)

            % Die Rueckgabe geht von einem Misserfolg aus. Wird eine Region
            % of Interrest gefunden, dann ist Success true und die ROIMask
            % enthaelt den Bildausschmitt als weissen Kasten. Die Zeilen
            % und Spalten Indexe im Vergleich zum Originalbild werden dann
            % auch gesetzt. Das ROISearchImage ist die verkleinerte oder in
            % seltenen Faellen auch die vergroesserte Version des
            % uebergebenen Bildes.
            Result.Success = false;
            Result.ROIMask = false(size(this.OriginalImage));
            Result.ROIImage = 0;
            Result.ROIRows = [];
            Result.ROIColumns = [];
            Result.ROISearchImage = this.WorkingImage;

            % Es folgt die Definition der Variable der Rueckgabestruktur,
            % die den Byte verbrauch der Objekte innerhalb dieses Objekts
            % speichert.
            Result.InternalBytesUsed = 0;

            % Es wirde eine adapive Histogrammequalisierung agewandt, falls
            % das Histogramm nur einen Peak hat.
            %EnhancedImage = Enhance(this.WorkingImage);

            % Das Arbeitsbild wird vor der region of interest suche von der
            % ungleichverteilten Beleuchtung befreit. Das Verfahren wird
            % fuer die Bereichssuche durchgefuehrt.
            Deillum = DEILLUM(this.WorkingImage, this.SearchHeight, 'smooth');
            %Deillum = DEILLUM(EnhancedImage, this.SearchHeight, 'morph');
            DeilluminatedImage = Deillum.Run();
            DeillumToStruct =  builtin('struct', Deillum); %#ok<NASGU>
            DeillumBytesUsed = whos('DeillumToStruct', 'bytes');
            DeillumBytesUsed = DeillumBytesUsed.bytes;
            clear DeillumToStruct;
            clear Deillum;

            % Es wirde eine adapive Histogrammequalisierung agewandt, falls
            % das Histogramm nur einen Peak hat.
            %EnhancedImage = Enhance(DeilluminatedImage);

            % Das Bild wird vor der region of interest suche von einer
            % unguenstigen, starken Schliffspuren befreit. Das Verfahren
            % wird fuer die Bereichsuche durchgefuhert.
            Degrind = DEGRIND(DeilluminatedImage, this.SearchHeight);
            %Degrind = DEGRIND(EnhancedImage, this.SearchHeight);
            DegrindedImage = Degrind.Run();
            DegrindToStruct =  builtin('struct', Degrind); %#ok<NASGU>
            DegrindBytesUsed = whos('DegrindToStruct', 'bytes');
            DegrindBytesUsed = DegrindBytesUsed.bytes;
            clear DegrindToStruct;
```

```
            clear Degrind;

            % Die Kantenmaske muss neu berechnet werden.
            %imshow(DegrindedImage);
            %waitforbuttonpress;
            this.WorkingImage = DegrindedImage;
            this.BuildEdgeMask();

            % Es wird der ganz normale Dougman Algorithmus ueber die Run
            % Methode der uebergeordneten Klasse gestartet. Falls das
            % Ergebnis einen Durchmesser von 0 hat, ist irgendetwas schief
            % gelaufen. Ansonsten wird die ROI Maske erzeugt.
            ParentResult = Run@TM(this);
            if(ParentResult.Diameter == 0)
                fprintf('ROI::Run: region of interest search failed for unknown reason↙
...\n');
                return;
            else

                % Dem gefundenen Kreis wird ein Kasten umgeschrieben.
                % BoundingBox enthalt die Start-Zeilen/Spalten, sowie die
                % Groesse des Kastens. Die Umrechnung in
                % Stop-Zeilen/Spalten kann leicht durch Addition erfolgen.
                % ceil berechnet die naechste hohere ganze Zahl.
                ROI = regionprops(ParentResult.CircleBoxMask, 'boundingbox');
                BoundingBox = ROI.BoundingBox;
                BoundingBox = ceil(BoundingBox);
                BoundingBox(1, 3) = BoundingBox(1, 1) + BoundingBox(1, 3);
                BoundingBox(1, 4) = BoundingBox(1, 2) + BoundingBox(1, 4);

                % Die BoundingBox wird etwas vergroessert. Die Variable
                % ROI.ExtendROI bestimmt um weiviele Zeilen/Spalten jeweils
                % vergroessert wird. Es muss geprueft werden ob der
                % Bildbereich nicht ueberschritten wurde. - Gegebenfalls
                % werden die maximal erlaubten Werte gesetzt.
                BoundingBox(1, 1) = BoundingBox(1, 1) - this.ExtendROI;
                BoundingBox(1, 2) = BoundingBox(1, 2) - this.ExtendROI;
                BoundingBox(1, 3) = BoundingBox(1, 3) + this.ExtendROI;
                BoundingBox(1, 4) = BoundingBox(1, 4) + this.ExtendROI;
                if(BoundingBox(1, 1) < 1), BoundingBox(1, 1) = 1; end
                if(BoundingBox(1, 2) < 1), BoundingBox(1, 2) = 1; end
                if(BoundingBox(1, 3) > size(this.WorkingImage, 2)), BoundingBox(1, 3) =↙
size(this.WorkingImage, 2); end
                if(BoundingBox(1, 4) > size(this.WorkingImage, 1)), BoundingBox(1, 4) =↙
size(this.WorkingImage, 1); end

                % Es wird eine kleine Region of Intereset Maske erzeugt,
                % die Groesse dieser Maske ist gleich wie die Groesse von
                % this.WorkingImage. this.WorkingImage ist geerbt. Diese
                % Maske wird dann auf Groesse des Originals gebracht.
                SmallROIMask = false(size(this.WorkingImage));
                SmallROIMask(BoundingBox(1, 2):1:BoundingBox(1, 4), BoundingBox(1, 1):↙
1:BoundingBox(1, 3)) = true;
                ROIMask = imresize(SmallROIMask, size(this.OriginalImage), 'nearest');
```

```
                % Jetzt koennen die Ergebnisse in Result abgelegt werden.
                % Die Suche nach der region of interrest ist beendet.
                [ROIRows, ROIColumns] = find(ROIMask == true);
                ROIRows = min(ROIRows):1:max(ROIRows);
                ROIColumns = min(ROIColumns):1:max(ROIColumns);
                Result.Success = true;
                Result.ROIMask = ROIMask;
                Result.ROIImage = this.OriginalImage(ROIRows, ROIColumns);
                Result.ROIRows = ROIRows;
                Result.ROIColumns = ROIColumns;

                Result.InternalBytesUsed = DeillumBytesUsed + DegrindBytesUsed;

            end

        end

    end

end
```

```
warning off all;
%clear all;
clc;

InputDirectory = '..\images\';
OutputDirectory = '.\results\';

FileHandle = fopen('time.txt', 'w+');
ResultsFileHandle = fopen('result_list.txt','w+');
fprintf(ResultsFileHandle, '# Result list, v1');

for i = 1:1:96
    bNR=(i-1)+1000;
    FileName = sprintf('B%d.png', bNR);
    Image = imread([InputDirectory '\' FileName]);

    a = D47(Image, 0.35, 0.5);
    tic;
    Result = a.Run;
    ElapsedTime = toc;
    aToStruct = builtin('struct', a);
    aBytesUsed = whos('aToStruct', 'bytes');
    aBytesUsed = aBytesUsed.bytes;
    clear a;
    x=Result.CenterColumn;
    y=Result.CenterRow;
    r=round(Result.Diameter / 2);
    BytesUsed = Result.InternalBytesUsed + aBytesUsed;
    fprintf(FileHandle, '%s\t%f\t%d\t%d\t%d\n', FileName, ElapsedTime, BytesUsed,↙
Result.InternalROISize(1, 1), Result.InternalROISize(1, 2));

    fprintf(ResultsFileHandle, '\r\nB%d.png %d/%d/%d', bNR, x, y, r);

    imwrite(Result.CircleOutlineImage, sprintf ('%s B%d.png', OutputDirectory, bNR));

end

fclose(FileHandle);
fclose(ResultsFileHandle);

%[~, ~, Channels] = size(Image);
%if(Channels > 1), Image = rgb2gray(Image); end
%tic;
%Result = a.Run;
%Time = toc;
%ClassToStruct = builtin('struct', a);
%UsedBytes = whos('ClassToStruct', 'bytes');
%UsedBytes = UsedBytes.bytes;
%FileHandle = fopen('time.txt','a+');
%fprintf(FileHandle, '%s\t%f\t%d\n', FileName, Time, UsedBytes);
%fclose(FileHandle);
%f = figure;
%subplot(2, 2, 1), imshow(Result.ROIMask);
%subplot(2, 2, 2), imshow(Result.ROIImage), title(sprintf('%dx%d', size(Result.↙
ROIImage, 1), size(Result.ROIImage, 2)));
```

```
%subplot(2, 2, 3), imshow(Result.ROISearchImage);
%subplot(2, 2, 4), imshow(Image);
%imshow(Result.CircleOutlineImage);
%waitforbuttonpress;
%close(f);
%clear a;
```

```
classdef BHAM < hgsetget
    %BHAM BRESENHAM CORE ALGORITHM
    %   bresenham core algorithm



    methods(Access = public, Static = true)

        % Diese Methode zeichnet einen Pixelkreis nach dem Bresenham
        % Algorithmus in eine Matrix. Es kann die Dicke des Kreises, sowie
        % ein Abstand zum Kreisrand angegeben werden. Die Dicke des Kreises
        % waechst nach innen. Eine Dicke von 0 ergibt einen Kreis von einem
        % Pixel. Ein Padding von 0 bedeutet der Kreis befindet sich an
        % allen Seiten direkt am Matrixrand.
        function [CircleMask] = BresenhamCircleMask(Diameter, Thickness, Padding)

            % Alle Parameter muessen Skalare sein.
            assert(isscalar(Diameter), 'BHAM::BresenhamCircleMask: Der angegebene↙
Durchmesser (Argument 1) ist kein Skalar.');
            assert(isscalar(Thickness), 'BHAM::BresenhamCircleMask: Die angegebene↙
Dicke (Argument 2) ist kein Skalar.');
            assert(isscalar(Padding), 'BHAM::BresenhamCircleMask: Der angegebene Rand↙
(Argument 3) ist kein Skalar.');

            % Erlaubte Wertebereiche pruefen.
            assert(round(Diameter) == Diameter, 'BHAM::BresenhamCircleMask: Der↙
Durchmesser (Argument 1) muss positiv und ganzzahlig und nicht 0 sein.');
            assert(Diameter > 2, 'BHAM::BresenhamCircleMask: Der Durchmesser (Argument↙
1) muss groesser als 2 sein.');
            assert(logical(bitget(Diameter, 1)), 'BHAM::BresenhamCircleMask: Der↙
Durchmesser (Argument 1) muss ungerade sein.');
            assert((round(Thickness) == Thickness) ||  (Thickness == 0), 'BHAM::↙
BresenhamCircleMask: Die Dicke (Argument 2) muss positiv und ganzzahlig oder 0 sein.');
            assert((round(Padding) == Padding) || (Padding == 0),  'BHAM::↙
BresenhamCircleMask: Das Polster (Argument 3) muss positiv und ganzzahlig oder 0↙
sein.');

            % Es wird noch ueberprueft, ob durch den angegebenen Dickenwert
            % ein unzulassiger Durchmesser entsteht.
            InnerDiameter = Diameter - 2 * Thickness;
            assert((InnerDiameter) > 2, 'BHAM::BresenhamCircleMask: Der Durchmesser↙
(Argument 1) und die Dicke (Argument 2) sind nicht kompatibel.');

            % Es werden die Masken fuer den inneren und den ausseren Kreis
            % erzeugt. Dabei wird das Polster (Padding) beruecksichtigt.
            % Die Masken fuer beide Kreise sind gleich gross.
            MaskSize = Diameter + 2 * Padding;
            OuterCircleMask = false(MaskSize, MaskSize);
            InnerCircleMask = OuterCircleMask;

            % Es werden nun die jeweiligen Kreise in die Masken
            % gespeichert. Dabei muss der Schreibbereich mit Zeilen und
            % Spalten angegeben werden. Falls die Kreisdicke 0 ist, koennen
            % weitere Berechnungen fuer den Innenkreis entfallen.
            OuterCircleMask((Padding + 1):(Padding + Diameter), (Padding + 1):(Padding↙
```

```
+ Diameter)) = BHAM.BresenhamCore(Diameter);
            if(Thickness == 0)
                CircleMask = OuterCircleMask;
                return;
            end
            InnerPadding = Padding + Thickness;
            InnerCircleMask((InnerPadding + 1):(InnerPadding + InnerDiameter),↙
(InnerPadding + 1):(InnerPadding + InnerDiameter)) = BHAM.BresenhamCore(InnerDiameter);

            % Der Innenbereich beider Masken wird gefuellt. Im Falle der
            % gefuellten Innenmaske wird der eigentliche Kreis mit der xor
            % Operation wider entfernt. Somit bleibt dieser Kreis dann beim
            % Bilden der Differenz zwischen Aussenkreis und Innenkreis
            % uebrig. Die Differnz ist wider eine xor Operation.
            InnerCircleMask = xor(InnerCircleMask, imfill(InnerCircleMask, 'holes'));
            OuterCircleMask = imfill(OuterCircleMask, 'holes');
            CircleMask = xor(InnerCircleMask, OuterCircleMask);

        end



        % Diese Funktion zeichnet einen Pixelkreis in einen Kasten. Es ist
        % nicht von Bedeutung, ob der Kreismittelpunkt innerhalb oder
        % ausserhlb des Kreises liegt. Die Funktion gibt eine Maske mit
        % logischen Werten zurueck.
        function [CircleBoxMask] = BresenhamCircleBoxMask(Diameter, Thickness,↙
CenterCoordinates, BoxExtents)

            % Es muessen nur noch die beiden letzen Parameter auf
            % Plausibilitaet geprueft werden. Die ersten beiden werden
            % durch eine andere Methode dieser Klasse geprueft.
            assert(isvector(CenterCoordinates) && (size(CenterCoordinates, 2) == 2) &&↙
(size(CenterCoordinates, 1) == 1), 'BHAM::BresenhamCircleBoxMask: Die↙
Mittelpunktskoordinaten (Argument 3) muessen in einem Zeilenvektor mit zwei Elementen↙
angegeben werden.');
            assert(all(round(CenterCoordinates) == CenterCoordinates), 'BHAM::↙
BresenhamCircleBoxMask: Die Mittelpunktskoordinaten (Argument 3) muessen ganze Zahlen↙
sein.');
            assert(isvector(BoxExtents) && (size(BoxExtents, 2) == 2) && (size↙
(BoxExtents, 1) == 1), 'BHAM::BresenhamCircleBoxMask: Die Kastengroesse (Argument 4)↙
muss in einem Zeilenvektor mit zwei Elementen angegeben werden.');
            assert(all(BoxExtents > [0 0]) && all(round(BoxExtents) == BoxExtents),↙
'BHAM::BresenhamCircleBoxMask: Die Kastenhoehe und Kastenbreite (Argument 4) muessem↙
positive ganze Zahlen ungleich 0 sein.');

            % Es wird die Kreismaske berechnet. Ob die Argumente zulaessig
            % sind wird von BHAM::BresenhamCircleMask getestet. Bei nicht
            % assert der nachfolgenden Funktion kann also probelemlost der
            % Radius berechnet werden.
            CircleMask = BHAM.BresenhamCircleMask(Diameter, Thickness, 0);
            Radius = (Diameter - 1)/2;

            % Der Kasten wird erstellt. Die Groesse wurde in einem
            % Zeilenvektor an die Funktion uebergen.
```

```
            CircleBoxMask = false(BoxExtents(1, 1), BoxExtents(1, 2));

            % Die zu schriebenden Zeilen/Spalten im Kasten werden
            % ermittelt. Die Anfangs und Endkoordinatene ergeben sich aus
            % dem Radiusvektor und den Mittelpunktskoordinaten.
            WriteStart = CenterCoordinates - [Radius, Radius];
            WriteStop = CenterCoordinates + [Radius, Radius];
            WriteAffectedRows = WriteStart(1, 1):1:WriteStop(1, 1);
            WriteExistingRows = 1:1:BoxExtents(1, 1);
            WriteToRows = intersect(WriteAffectedRows, WriteExistingRows);
            WriteAffectedColumns = WriteStart(1, 2):1:WriteStop(1, 2);
            WriteExistingColumns = 1:1:BoxExtents(1, 2);
            WriteToColumns = intersect(WriteAffectedColumns, WriteExistingColumns);

            % Die zu lesenden Zeilen/Spalten im Kasten werden
            % ermittelt.
            ReadStart = [Radius + 2, Radius + 2] - CenterCoordinates;
            ReadStop = [Radius + 1, Radius + 1] + BoxExtents - CenterCoordinates;
            ReadAffectedRows = ReadStart(1, 1):1:ReadStop(1, 1);
            ReadExistingRows = 1:1:Diameter;
            ReadFromRows = intersect(ReadAffectedRows, ReadExistingRows);
            ReadAffectedColumns = ReadStart(1, 2):1:ReadStop(1, 2);
            ReadExistingColumns = 1:1:Diameter;
            ReadFromColumns = intersect(ReadAffectedColumns, ReadExistingColumns);

            % Die Daten werden aus der Kreismaske gelesen und in die
            % Kastenmaske geschrieben.
            CircleBoxMask(WriteToRows, WriteToColumns) = CircleMask(ReadFromRows,↙
ReadFromColumns);

        end

    end



    methods(Access = private, Static = true)

        % Implementierung des von Jack Bresenham 1962 entworfenen
        % Pixelkreisalgorithmus. Der Durchmesser muss ungerade sein. Die
        % Zulaessigkeit des einzigen Arguments wird von einer anderen
        % Methode der Klasse ueberprueft. Das Rueckgabewert ist eine
        % logische Maske, in der alle Pixel des Kreises den Wert true und
        % alle anderen Pixel den Wert false annehmen.
        function [CircleMask] = BresenhamCore(Diameter)

            % Nachfolgend werden die Startwerte der Schleife berechnet. Mit
            % Hilfe der Entscheidungsvariable d werden die richtigen Punkte
            % des Kreises gewaehlt. Der geniale Grundgedanke des
            % Algorithmus liegt in der iterativen Berechnung der
            % Entscheidungsvariable. Somit sind nur die Rechenoperationen
            % Addition und Subtraktion noetig.
            Radius = (Diameter - 1)/2;
            CircleMask = false(Diameter, Diameter);
            y = 1; % Spalten Index
```

```
            x = 1 + Radius; % Zeilen Index
            d = 1 - Radius; % Entscheidungsvariable

            % kreispunkte erzeugen
            CircleMask(Radius + x, Radius + y) = true;
            while(x > y)
                if(d < 0)
                    d = d + 2 * y + 3;
                    y = y + 1;
                else
                    d = d + 2 * (y - x) + 5;
                    y = y + 1;
                    x = x - 1;
                end
                CircleMask(Radius + x, Radius + y) = true;
            end

            % Der Bresenham Algorithmus zeichnet nur ein 45 Grad Segment
            % eines Kreises. Dieses Segment kann durch Spiegloperationen zu
            % einem vollstandigen Kreis kombiniert werden. Nachfolgend wird
            % eine Spiegel-Matrix erzeugt.
            IdentityMatrix = logical(eye(Diameter, Diameter));
            MirrorMatrix = rot90(IdentityMatrix) | IdentityMatrix;
            MirrorMatrix(1:Radius,:) = false;

            % Um Matrizenmultiplikationen durchzufuehren benoetigt Matlab
            % den Datentyp double. CircleMask wird entsprechend
            % konvertiert. Am Ende erfolgt Rueckwandlung in logical.
            CircleMask = double(CircleMask);
            CircleMask = CircleMask * double(MirrorMatrix);
            CircleMask = rot90(CircleMask);
            CircleMask = CircleMask * double(MirrorMatrix);
            CircleMask = logical(CircleMask) | logical(rot90(CircleMask));

        end

    end


end
```

```
classdef DMAN < BHAM
    %DMAN DAUGMAN INTEGRO DIFFERNTIAL OPERATOR CORE ALGORITHM
    %   daugman integro differential operator core algorithm



    properties(Constant = true)

        % Nachfolgend werden die Standardbeschraenkungen fuer Bildhohe und
        % Bildbreite definiert. Diese Werte sind obere Grenzen. Der
        % Bruchteil des kleinsten Radius an der kleineren Seite ist nach
        % unten beschraenkt, Werte groesser als 1 ergeben auch keinen Sinn.
        % Die MinimumStartDiameterFraction muss mit den kleinsten
        % Bildgroessen von 24 konsitent sein und sollte daher nicht
        % kleiner als 0.3 werden.
        MaximumHeight = 640;
        MaximumWidth = 640;
        MinimumStartDiameterFraction = 0.3;

    end



    properties(Access = protected)

        % Es folgt die Definition der Variable fuer das zu bearbeitende
        % Bild. Mehrere Methoden greifen darauf zu. Die Variable wird vom
        % Konstruktor gesetzt.
        WorkingImage = 0;

    end



    properties(Access = private)

        % Es folgt der Durchmesserstartwert, als Anteil der kleineren
        % Seite. Die Variable wird vom Konstruktor gesetzt.
        StartDiameterFraction = 0;

        % Mehrere Methoden greiffen auf diese Variablen zu, gesetzt werden
        % sie von DMAN::LoadPathIntegralMasks und DMAN::ProducePathIntegralMasks
        PathIntegralMasks = 0;
        PathIntegralsStack = [];
        DerivativesStack = [];
        StartDiameter = 0;
        EndDiameter = 0;

    end



    methods

        % Konstrukter der DMAN Klasse. Dem Konstruktor muss das zu
```

```
        % bearbeitende Bild und die Verarbeitungsgroesse des Bildes
        % uebergeben werden. Die Groesse darf die MaximumHeight nicht
        % ueberschrieten.
        function this = DMAN(Image, Height, StartDiameterFraction)

            % Das Bild muss ein Graustufenbild sein. Ein Bild mit drei
            % Farbkannaelen wird nicht akzeptiert. Das Bild muss mindestens
            % 24 mal 24 Groesse haben.
            assert(ndims(Image) == 2, 'DMAN::DMAN: Das Bild (Argument 1) muss ein↙
Graustufenbild sein.');
            assert(all(size(Image) > [23 23]), 'DMAN::DMAN: Das Bild (Argument 1) muss↙
mindestens eine Groesse von 24x24 haben.');

            % Die angegebene Hoehe darf die maximale Hohe nicht
            % ueberschreiten und darf nicht kleiner als 24 sein.
            assert(isscalar(Height), 'DMAN::DMAN: Die Bildhoehe (Argument 1) muss ein↙
Skalar sein sein.');
            assert((round(Height) == Height) && (Height < (DMAN.MaximumHeight + 1)) &&↙
(Height > 23), sprintf('DMAN::DMAN: Die Bildhoehe muss ganzzahlig, mindestens 24 und↙
hoechstens %d sein.', DMAN.MaximumHeight));

            % Das Bild wird verkleinert und es wird geprueft, ob die maximal
            % zulaessige Breite nicht ueberschritten wird.
            Image = imresize(Image, [Height NaN], 'bicubic');
            assert((size(Image, 2) < (DMAN.MaximumWidth + 1)) && (size(Image, 2) > 23),↙
sprintf('DMAN::DMAN: Die Bildbreite darf %d nicht ueberschreiten und nicht kleiner als↙
24 sein. Sie ist %d.', DMAN.MaximumWidth, size(Image, 2)));

            % Der Startwert des Durchmessers wird als Antiel der kleineren
            % Seite angegeben. Er darf nicht grosser als 1 und kleiner als
            % der minimale Wert sein.
            assert((StartDiameterFraction <= 1) && (StartDiameterFraction >= DMAN.↙
MinimumStartDiameterFraction), sprintf('DMAN::DMAN: Der Anteil des Anfangsdurchmessers↙
an der kleineren Seite muss zwischen 1 und %d liegen.', DMAN.↙
MinimumStartDiameterFraction));

            % Wenn alles in Ordnung ist, wird das Bild abgespeichert.
            this.WorkingImage = Image;
            this.StartDiameterFraction = StartDiameterFraction;

        end

    end



    methods(Access = public)

        % Startet den Algorithmus und gibt eine Struktur als Ergebnis
        % zurueck, darin sind die Mittelpunktsdaten, der Druchmesser, das
        % Bild und das Bild mit eingezeichnetem Kreis enthalten.
        function [Result] = Run(this)

            % Es wird versucht vorhandene Pfadintegralmasken zu laden.
            % Falls sie nicht gefunden werden muessen sie neu erstellt und
```

```
            % gespeichert werden.
            Loaded = this.LoadPathIntegralMasks();
            if(~Loaded)
                this.ProducePathIntegralMasks();
                this.SavePathIntegralMasks();
            end

            % Nun startet die eigentliche Berechnung der Pfadintegrale. Es
            % handelt sich um die Berechnung des Faltungsprodukts der Pfad-
            % Integral-Masken mit dem WorkingImage.
            this.CalculatePathIntegrals();

            % Fuer alle moeglichen Mittelpunkte im Bild werden die
            % numerischen Ableitungen nach den Pfadintegralergebnissen
            % berechnet. Die numerische Ableitung wird abschliessend durch
            % einen Gaussfilter geglaettet um moegliche Spruenge, die die
            % Maximumsuche stoeren koennten zu entfernen.
            this.CalculateDerivative();

            % Die Suche nach den globalen Maximalstellen kann nun gestartet
            % werden. Die Rueckgabe muss untersucht werden. Ist der
            % Durchmesser Null, dann war liegt ein Null WorkingImage vor,
            % oder es ist etwas unvorhergesehens passiert. Die Aussenlinie
            % des Kreises wird nur dann eingezeichnet, falls sie berechnbar
            % ist und auf das ganze Bild passt.
            Best = this.FindMaximumIndexes();

            % Die Ergebnisse ausgeben. Ein Durchmesser von Null deutet
            % einen nicht sinnvollen Kreisfund an. Ansonsten wird der
            % Umfang des Kreises eingezeichnet. Dazu wird der Bresenham
            % Algorithmus verwendet.
            Result.WorkingImage = this.WorkingImage;
            Result.CenterRow = Best.Row;
            Result.CenterColumn = Best.Column;
            if(Best.Diameter ~= 0)

                % Der beste Radius wird in der Rueckgabe Struktur
                % gespeichert. Dieser Durchmesser ist hier sicher >= 3.
                Result.Diameter = Best.Diameter;

                % Die Aussenlinie des Kreises muss noch in das Bild
                % eingezeichnet werden. Dazu wird der Bresenham-Algorithmus
                % verendet. Das Aussenlinienbild ist ein RGB Bild.
                CircleBoxMask = DMAN.BresenhamCircleBoxMask(Best.Diameter, 2, [Best.↙
Row, Best.Column], size(this.WorkingImage));
                Red = this.WorkingImage;
                Green = this.WorkingImage;
                Blue = this.WorkingImage;
                Red(CircleBoxMask) = 255;
                Green(CircleBoxMask) = 0;
                Blue(CircleBoxMask) = 0;
                Result.CircleBoxMask = CircleBoxMask;
                Result.CircleOutlineImage = cat(3, Red, Green, Blue);

            else
```

```
                Result.Diameter = 0;
                Result.CircleBoxMask = false(this.WorkingImage);
                Result.CircleOutlineImage = cat(3, this.WorkingImage, this.↙
WorkingImage, this.WorkingImage);
            end

        end

    end



    methods(Access = private)

        % Uberprueft ob es vorhandene Pfadintegralmasken im
        % Arbeitsverzeichnis gibt. Falls diese vorhanden sind werden sie
        % geladen und verwendet. Sind keine passenden Masken vorhanden,
        % welche geladen werden koennten ist loaded false.
        function [Loaded] = LoadPathIntegralMasks(this)

            % Es muss zunaechst berechnet werden, mit welchem Durchmesser
            % gestartet wird und welcher Durchmesser der Letzte ist. Diese
            % Daten identifizieren auch die gespeicherten Masken. Die
            % Durchmesser muessen immer ungerade sein. Der StartDurchmesser
            % muss um 2 tiefer angesetzt werden, da bei der Differenziation
            % ein Wert verlohren geht. Um nach der Differenziation eine
            % drei dimensionale Matrix sicher zustellen muss zwei
            % Durchmesser kleiner begonnen werden.
            SmallerSide = min(size(this.WorkingImage));
            this.StartDiameter = round(SmallerSide * this.StartDiameterFraction);
            this.StartDiameter = this.StartDiameter + double(~bitget(this.↙
StartDiameter, 1));
            this.StartDiameter = this.StartDiameter - 4;
            this.EndDiameter = SmallerSide;
            this.EndDiameter = this.EndDiameter - double(~bitget(this.EndDiameter, 1));

            % Es wird nun eine Array mit der benoetigten Groesse erstellt.
            % Der Array hat eine Zeile und die geforderte Anzahl an
            % Spalten. Die Groesse kann immer mit size abgefragt werden.
            this.PathIntegralMasks = cell(1, ((this.EndDiameter - this.StartDiameter)↙
/2) + 1);

            % Nachfolgend wird der Dateiname der gesuchten Maskendatei
            % zusammengestellt und danach die Datei geladen.
            FileName = sprintf('dman.%d.%d.mat', this.StartDiameter, this.EndDiameter);
            if(exist(FileName, 'file'))
                fprintf('DMAN::LoadPathIntegralMasks: mask file exists, loading path↙
integral masks from workspace ...\n');
                load(FileName, 'Masks');
                this.PathIntegralMasks = Masks;
                clear Masks;
                Loaded = true;
            else
                fprintf('DMAN::LoadPathIntegralMasks: mask file does not exists, cannot↙
load something ...\n');
```

```
                Loaded = false;
            end

        end


        % Diese Methode erzeugt die Masken zur Berechnnung des
        % Pfadintegrals. Die Funktion BHAM::BresenhamCircleMask wird dazu
        % verwendet. DMAN::LoadPathIntegralMasks muss vorher gelaufen sein.
        function [] = ProducePathIntegralMasks(this)

            % Es wird nun ein Vektor mit allen Durchmessern erzeugt und die
            % entsprechende Kreismaske mit dem Bresenham Algorithmus
            % erstellt. DMAN::BresenhamCircleMask wurde von BHAM vererbt.
            Diameters = this.StartDiameter:2:this.EndDiameter;
            for i = 1:1:size(this.PathIntegralMasks, 2)

                % Maske mit dem Bresenham Algorithmus erzeugen.
                PathIntegralMask = DMAN.BresenhamCircleMask(Diameters(1, i), 0, 0);

                % Maske und Daten der Maske werden abgespeichert.
                this.PathIntegralMasks{1, i}.Circumference  = sum(sum(double↙
(PathIntegralMask)));
                this.PathIntegralMasks{1, i}.Mask = double(PathIntegralMask);
                this.PathIntegralMasks{1, i}.Diameter = Diameters(1, i);
                this.PathIntegralMasks{1, i}.Radius = (Diameters(1, i) - 1)/2;

                % Es wird ausgegeben, welche von wievielen Masken gerade
                % erstellt wird. Die gesamte Zahl ist die Grosse des vorher
                % erstellten Arrays this.PathIntegralMasks.
                fprintf('DMAN::ProducePathIntegralMasks: producing mask %d/%d ...\n',↙
i, size(this.PathIntegralMasks, 2));

            end

        end



        % Diese Funktion speichert die Pfad Integral Masken in einer Datei
        % ab um sie spaeter schnell laden zu koennen. Diese Funktion
        % benoetigt ein vorheriges Laufen von
        % DMAN::ProducePathIntegralMasks.
        function [] = SavePathIntegralMasks(this)

            % Nun wird der Dateiname zusammengestellt. Falls die Datei im
            % Arbeitsverzeichnis bereits existiert, muss sie nicht extra
            % neu gespeichert werden.
            FileName = sprintf('dman.%d.%d.mat', this.StartDiameter, this.EndDiameter);
            if(exist(FileName, 'file'))
                fprintf('DMAN::SavePathIntegralMasks: mask file exists, no need to save↙
masks ...\n');
            else
                fprintf('DMAN::SavePathIntegralMasks: writing masks to file ...\n');
                Masks = this.PathIntegralMasks;
```

```
                save(FileName, 'Masks');
                clear Masks;
            end

        end



        % Berechnet die Pfadintegrale fuer alle moegliechen Mittelpunkte
        % des Bildes und alle moeglichen Radien bzw. Durchmesser. - Das ist
        % eine Faltung des Bildes mit allen Pfad-Integral-Masken, die im
        % Speicher sind. Die Funktion benoetigt das vorherige Laufen von
        % DMAN::ProducePathIntegralMasks.
        function [] = CalculatePathIntegrals(this)

            % Zunaechst wird ein Array der benoetigten Groesse erstellt. In
            % diesem Integral werden alle Pfadintegrale gespeichert. Des
            % Weiteren wird eine Matrix mit Bildgroesse erstellt, um allen
            % Pfadintegralmatrizen dieselbe Groesse zu geben, auch wenn der
            % zulassige Bereich, bedingt durch die Maskengroesse kleiner
            % als das Bild selbst ist.
            this.PathIntegralsStack = zeros(size(this.WorkingImage, 1), size(this.↙
WorkingImage, 2), size(this.PathIntegralMasks, 2));
            WholeRegion = zeros(size(this.WorkingImage));
            for i = 1:1:size(this.PathIntegralMasks, 2)

                % Das Programm gibt aus welches Pfadintegral von wievielen
                % es gerade berechnet. - Dazu die gesamte Anzahl.
                fprintf('DMAN::CalculatePathIntegrals: calculation path integral %d/%d↙
...\n', i, size(this.PathIntegralMasks, 2));

                % Das Faltungsprodukt wird nun berechnet und jedes Ergebnis
                % durch den Umfang des Kreispfades normiert.
                ValidRegion = conv2(double(this.WorkingImage), double(this.↙
PathIntegralMasks{i}.Mask), 'valid');
                ValidRegion = ValidRegion./this.PathIntegralMasks{1, i}.Circumference;

                % Nachfolgend wird berechnet, welchen Bereich die
                % ValidRegion innerhalb der WholeRegion ausfuellt.
                % Entsprechend der brechneten Zeilten und Splaten werden
                % die Daten zugewiesen. Das Faltungsproduktergebis hat
                % somit immer die Groesse des Originalbildes obwohl es nur
                % fuer einen kleineren Bereich berechnet werden kann.
                Radius = this.PathIntegralMasks{i}.Radius;
                Rows = (Radius + 1):(Radius + size(ValidRegion, 1));
                Columns = (Radius + 1):(Radius + size(ValidRegion, 2));
                WholeRegion(Rows, Columns) = ValidRegion;

                % Abschliessend werden die Daten in der Eigenschaft der
                % Klasse abgespeichert. - Die DMAN::CalculateDerivatives
                % Methode kann die Daten anschliessend weiterverarbeiten.
                this.PathIntegralsStack(:, :, i) = WholeRegion;

                % Das Faltungsprodukt hat einen eingschraenketen
                % Gueltigkeitsbereich, der kleiner als das ganze Bild ist.
```

```
                % Dieser Bereich wird kleiner umso groesser die Maske
                % wird mit der die Faltung statt findet.
                WholeRegion(:, :) = this.PathIntegralsStack(:, :, i);

            end

        end



        % Diese Methode berechnet die numerische Ableitung der
        % Pfadintegrale fuer jeden moeglichen Mittelpunkt im Bild. Die
        % Funktion benoetigt ein vorheriges Laufen von
        % DMAN::CalculatePathIntegrals.
        function [] = CalculateDerivative(this)

            % Nachfolgend wird ein Array fuer die Ableitungen definiert.
            % Cuboids ist eine drei dimensionale Matrix, in der alle
            % Pfad-Integrale uebereinander gestapelt werden. In dieser
            % Anordnung ist Matlab im Stande die numerische Ableitung
            % relativ schnell zu brechnen.
            this.DerivativesStack = zeros(size(this.WorkingImage, 1), size(this.↙
WorkingImage, 2), (size(this.PathIntegralMasks, 2) - 1));

            % Die momentan durchgefuehrte Aktion wird in der Konsole
            % oder auf der Standardausgabe angezeigt.
            fprintf('DMAN::CalculateDerivative: ableitung in radiale richtung berechnen↙
...\n');

            % Nachfolgend findet die Ermittlung der numerischen
            % Ableitung statt. diff verlangt als zweiten Parameter die
            % Stufe der Ableitung und als dritten Parameter die
            % Dimension entlang der die Ableitung erfolgen soll.
            this.DerivativesStack = diff(this.PathIntegralsStack, 1, 3);
            this.DerivativesStack = abs(this.DerivativesStack);

            % Die momentan durchgefuehrte Aktion wird in der Konsole
            % oder auf der Standardausgabe angezeigt.
            fprintf('DMAN::CalculateDerivative: applying gaussian low pass filter ...↙
\n');

            % Abschliessend werden die Ableitungen an den moeglichen
            % Mittelpunktsstellen mit einem Gaussschen Tiefpassfilter
            % geglaettet. Das hilft der anschliessenden Maximumsuche.
            % Der Gaussfilter ist konstant, er sollte eigentlich mit
            % dem Radius variiren.
            GaussianLowPass = fspecial('gaussian', [1, 5], 1);
            this.DerivativesStack = permute(this.DerivativesStack, [1, 3, 2]);
            this.DerivativesStack = convn(this.DerivativesStack, GaussianLowPass,↙
'same');
            this.DerivativesStack = permute(this.DerivativesStack, [1, 3, 2]);

        end
```

```
        % Diese Methode sucht nach den globalen Maximalstellen innerhalb
        % der Ableitung in radiale Richtung. Falls mehrere konzentrische
        % Kreise vorliegen, dann wird immer der staerker ausgepraegte
        % vorgezogen. Dieses Verhalten ist so verlangt.
        function [Best] = FindMaximumIndexes(this)

            % Der Rueckgabewert ist eine Struktur mit den Daten der besten
            % Maximalstelle ueber alle moeglichen Mittelpunkte. Wenn das
            % WorkingImage tatsaechlich nur aus Nullen besteht wird das
            % Maximum als Null zurueckgegeben. Der Mittelpunkt ist dann im
            % linken oberen Eck und der Durchmesser des Kreises ist Null.
            Best.Maximum = 0;
            Best.Diameter = 0;
            Best.Row = 1;
            Best.Column = 1;

            % Nachfolgend werden die globalen Maxima der dritten
            % Dimension ermittelt, also die Maxima die in Radiale
            % Richtung auftreten. Danach werden der hoechste Wert eines
            % bestimmten Mittelpunktes ausgewahlt
            [ThirdMaxima, ThirdIndexes] = max(this.DerivativesStack, [], 3);
            [ColumnMaxima, RowIndexes] = max(ThirdMaxima, [], 1);
            [Maximum, ColumnIndex] = max(ColumnMaxima);

            % Es ist moeglich, dass konzentrische Kreise auftreten, in
            % diesem Fall muss am selben Mittelpunkt nach einer hohen
            % Antwort gesucht werden.
            OtherMaxima = this.DerivativesStack(RowIndexes(ColumnIndex), ColumnIndex,↙
ThirdIndexes(RowIndexes(ColumnIndex), ColumnIndex):end);
            OtherMaximaFraction = OtherMaxima./Maximum;
            OtherMaximaIndexes = (OtherMaximaFraction >= 1);
            if(any(OtherMaximaIndexes))
                fprintf('DMAN::FindMaximumIndexes: %d other maxima are present ...\n',↙
numel(OtherMaximaIndexes))
                Maximum = max(OtherMaxima(OtherMaximaIndexes));
            end

            % Das globale Maximum aller moeglichen Mittelpunkte und aller
            % moeglichen Radien wird als der gefundene Kreismittelpunkt
            % verwendet. Der Durchmesser des Ergbniskreises ist der
            % Anfangsdurchmesser der Suche plus den doppelten Radiuswert,
            % an dem das globale Maximum auftritt.
            Best.Maximum = Maximum;
            Row = RowIndexes(ColumnIndex);
            Column = ColumnIndex;
            Best.Row = Row;
            Best.Column = Column;
            Best.Diameter = this.PathIntegralMasks{1}.Diameter + 2 * ThirdIndexes(Row,↙
Column);

            % Die besten Werte werden auf der Standardausgabe angezeigt.
            fprintf('DMAN::FindMaximumIndexes: best diameter is %d at row %d and column↙
%d ...\n',  Best.Diameter, Best.Row, Best.Column);
```

```
        end

    end

end
```

```
classdef D47 < hgsetget
    %D47 DAUGMAN INTEGRO DIFFERENTIAL OPERATOR BASED ROI AND DAUGMAN INTEGRO↙
DIFFERENTIAL OPERATOR BASED ALGORITHM
    %   daugman integro differential operator based roi and integro differential↙
operator based algorithm



    properties(Constant = true)

        % Innerhalb der Bereichssuche sollte der Startwert des
        % Kreisdurchmessers sehr klein sein. Der minimal zulassige Anteil
        % an der kleinern Bildseite ist 0.3. Um auch die kleinsten Kreise
        % zu finden wird dieser Wert auf 0.3 gesetzt.
        MinimumROISearchStartDiameterFraction = 0.3;

        % Wenn das ausgeschnittene Bild durchsucht wird darf der kleinste zu
        % suchende Durchmesser einen Anteil von 0.4 nicht unterschreiten,
        % da sonst die Faltungsberechnung zu aufwendig wird.
        MinimumFoundROIStartDiameterFraction = 0.4;

    end



    properties(Access = private)

        % Es folgt die Definition des Seicherplatzes fuer das Bild. Die
        % Variable wird vom Konstruktor oder der entsprechenden set Methode
        % gesetzt.
        OriginalImage = [];

        % Es folgt die Definition des kleinst moegliche Anteils des
        % Kreisdurchmessers an der kleineren Bildseite fuer die Region of
        % Interest Suche und fuer die anschliessende Suche des Kreises
        % innerhalb der Region of Interest.
        ROISearchStartDiameterFraction = 0;
        FoundROIStartDiameterFraction = 0;

    end



    properties(Access = private, Dependent = true)

        GrayscaleImage = [];

    end



    methods

        function this = D47(OriginalImage, ROISearchStartDiameterFraction,↙
FoundROIStartDiameterFraction)
```

```
            % An den Konstuktor uebergebenes Bild in der Memeber-Variable
            % abspeichern.
            set(this, 'OriginalImage', OriginalImage);

            % Die jeweiligen Startwert in den Membervariablen abspeichern.
            set(this, 'ROISearchStartDiameterFraction',↙
ROISearchStartDiameterFraction);
            set(this, 'FoundROIStartDiameterFraction', FoundROIStartDiameterFraction);

        end



        function this = set.OriginalImage(this, Value)

            % Im nachfolgenden Programmtext wird die eingabe auf
            % Gueltigkeit geprueft. Es kann sich um ein Graustufenbild oder
            % ein RGB-Bild handeln.
            assert((ndims(Value) == 2) || (ndims(Value) == 3), 'D47::set.OriginalImage:↙
Das Bild (Argument 1) muss ein Graustufenbild oder ein RGB Bild sein.');
            assert(((size(Value, 3) == 1) || (size(Value, 3) == 3)) && (all([size↙
(Value, 1) size(Value, 2)] > [23 23])), 'D47::D47: Das Bild (Argument 1) muss ein↙
Graustufenbild oder ein RGB Bild und mindestens 24x24 in der Groesse sein.');

            % Nach dem die Eingabe ueberprueft worden ist kann das Bild
            % gespeichert werden.
            this.OriginalImage = Value;

        end



        function this = set.ROISearchStartDiameterFraction(this, Value)

            % Die uebergebene Eigenschaft wird nachfolgend auf
            % plausibilitaet geprueft.
            assert(isscalar(Value), 'D47::setROISearchStartDiameterFraction: Die↙
Eigenschaft muss ein Skalar sein.');
            assert(isreal(Value), 'D47::setROISearchStartDiameterFraction: Die↙
Eigenschaft mus reell sein.');
            assert((D47.MinimumROISearchStartDiameterFraction <= Value) && (Value <=↙
1), sprintf('D47::setROISearchStartDiameterFraction: Die Eigenschaft liegt ausserhalb↙
des gueltigen Bereichs von [%d 1].', D47.MinimumROISearchStartDiameterFraction));

            % Falls der Wert korrekt ist wird er in der Eigenschaft der
            % Klasse abgespeichert.
            this.ROISearchStartDiameterFraction = Value;

        end



        function this = set.FoundROIStartDiameterFraction(this, Value)
```

```
            % Die uebergebene Eigenschaft wird nachfolgend auf
            % plausibilitaet geprueft.
            assert(isscalar(Value), 'D47::setFoundROIStartDiameterFraction: Die↙
Eigenschaft muss ein Skalar sein.');
            assert(isreal(Value), 'D47::setFoundROIStartDiameterFraction: Die↙
Eigenschaft mus reell sein.');
            assert((D47.MinimumFoundROIStartDiameterFraction <= Value) && (Value <= 1),↙
sprintf('D47::setFoundROIStartDiameterFraction: Die Eigenschaft liegt ausserhalb des↙
gueltigen Bereichs von [%d 1].', D47.MinimumFoundROIStartDiameterFraction));

            % Falls der Wert korrekt ist wird er in der Eigenschaft der
            % Klasse abgespeichert.
            this.FoundROIStartDiameterFraction = Value;

        end



        function Value = get.GrayscaleImage(this)

            % Falls das Bild mehrer Farbkanaele hat wird ein Graustufenbild
            % erzeugt und zurueckgegeben.
            if(size(this.OriginalImage, 3) == 3)
                Value = rgb2gray(this.OriginalImage);
            else
                Value = this.OriginalImage;
            end

        end

    end



    methods

        function [Result] = Run(this)

            % Im nachfolgenden Codeblock werden die Standardwerte der
            % Rueckgabevariable voreingestellt. Bei erfolgreicher
            % Bereichseinschraenkung werden die Werte entspraechend
            % angepassst.
            Result.Success = false;
            Result.OriginalImage = this.OriginalImage;
            Result.CenterRow = 1;
            Result.CenterColumn = 1;
            Result.Diameter = 0;
            Result.CircleOutlineImage = cat(3, get(this, 'GrayscaleImage'), get(this,↙
'GrayscaleImage'), get(this, 'GrayscaleImage'));
            Result.CircleBoxMask = false(size(this.OriginalImage, 1), size(this.↙
OriginalImage, 2));

            Result.InternalBytesUsed = 0;
            Result.InternalROISize = [0 0];
```

```
            % Es wird die Region Interest Suche mit dem Daugman
            % Algorithmus gestartet. Das Bild wird dabei signifikant
            % verkleinert.
            Roi = ROI(get(this, 'GrayscaleImage'), this.↙
ROISearchStartDiameterFraction);
            ROISearchResult = Roi.Run();

            % Der Speicherverbrauch der Region of Interest Klasse wird
            % ermittelt um ihn in der Rueckgabestruktur zu speicheren.
            RoiToStruct = builtin('struct', Roi);
            RoiBytesUsed = whos('RoiToStruct', 'bytes');
            RoiBytesUsed = RoiBytesUsed.bytes;
            clear Roi;
            clear RoiToStruct;

            % Falls die Region of Interest Suche fehlschlaegt, dann ist der
            % Durchmesser 0 und das Programm muss agbebrochen werdem.
            if(~ROISearchResult.Success)
                fprintf('D47::Run: unbekannter Fehler bei der Bereichseinschraenkung↙
...\n');
                return;
            else

                fprintf('D47::Run: region of interest gefunden, Kreisssuche starten ...↙
\n');

                % Falls die Bereichseinschraenkung erfolgreich verlaufen
                % ist wird der Daugman Algorthmus nochmals auf den
                % eingeschraenkten Bereich angewendet, dabei kann der
                % Region of Interest Bereich eine Groesse von 640x640
                % ueberschrieten, ist das der Fall muss auf diese Groesse
                % eingschraenkt werden.
                if(any(size(ROISearchResult.ROIImage) > [640 640]))

                    fprintf('D47::Run: region of interest exceeds maximum size allowed,↙
scaling down ...\n');

                    [~, MaximumSideIndex] = max(size(ROISearchResult.ROIImage));
                    ScaleDownVector = [NaN NaN];
                    ScaleDownVector(1, MaximumSideIndex) = 640;
                    ScaleDownROI = imresize(ROISearchResult.ROIImage, ScaleDownVector,↙
'bicubic');
                    Dman = DMAN(ScaleDownROI, size(ScaleDownROI, 1), this.↙
FoundROIStartDiameterFraction);
                    FoundROIResult = Dman.Run();
                    FoundROIResult.CircleOutlineImage = imresize(FoundROIResult.↙
CircleOutlineImage, size(ROISearchResult.ROIImage), 'bicubic');
                    FoundROIResult.CircleBoxMask = imresize(FoundROIResult.↙
CircleBoxMask, size(ROISearchResult.ROIImage), 'bicubic');
                    FoundROIResult.CenterRow = round(FoundROIResult.CenterRow * size↙
(ROISearchResult.ROIImage, 1)/size(ScaleDownROI, 1));
                    FoundROIResult.CenterColumn = round(FoundROIResult.CenterColumn *↙
size(ROISearchResult.ROIImage, 2)/size(ScaleDownROI, 2));
                    FoundROIResult.Diameter = round(FoundROIResult.Diameter * size↙
(ROISearchResult.ROIImage, 2)/size(ScaleDownROI, 2));
```

```
                else
                    ROIImage = ROISearchResult.ROIImage;
                    Dman = DMAN(ROIImage, size(ROISearchResult.ROIImage, 1), this.↙
FoundROIStartDiameterFraction);
                    FoundROIResult = Dman.Run();
                end

                % Der Speicherverbrauch der Daugman Klasse wird
                % ermittelt um ihn in der Rueckgabestruktur zu speicheren.
                DmanToStruct = builtin('struct', Dman); %#ok<NASGU>
                DmanBytesUsed = whos('DmanToStruct', 'bytes');
                DmanBytesUsed = DmanBytesUsed.bytes;
                clear Dman;
                clear DmanToStruct;

                % Das Endergebnis berechnen und in der Rueckgabevariable
                % abspeichern. Der Offset der Region of Interest muss dazu
                % addiert werden.
                Result.Success = true;
                Result.CenterRow = FoundROIResult.CenterRow + min(ROISearchResult.↙
ROIRows) - 1;
                Result.CenterColumn = FoundROIResult.CenterColumn + min↙
(ROISearchResult.ROIColumns) - 1;
                Result.Diameter = FoundROIResult.Diameter;
                Result.CircleOutlineImage(ROISearchResult.ROIRows, ROISearchResult.↙
ROIColumns, :) = FoundROIResult.CircleOutlineImage;
                Result.CircleBoxMask(ROISearchResult.ROIRows, ROISearchResult.↙
ROIColumns) = FoundROIResult.CircleBoxMask;

            end

            Result.InternalBytesUsed = RoiBytesUsed + ROISearchResult.InternalBytesUsed↙
+ DmanBytesUsed;
            Result.InternalROISize = [numel(ROISearchResult.ROIRows), numel↙
(ROISearchResult.ROIColumns)];

        end

    end

end
```

```
classdef DEGRIND < hgsetget
    %DEGRIND FOURIER BASED GRINDING REMOVE PROGRAM
    %   fourier based grinding remove program

    properties(Constant = true)

        % Nachfolgend werden die maximal zulaessigen Bildgroessen
        % festgelegt. Eine Ueberschreitung dieser Groessen kann in einer
        % nicht endenden Fouriertransformation enden.
        MaximumHeight = 640;
        MaximumWidth = 640;

        % Nachfolgend wird die Standardgroesse des Spektrum
        % Glaettungsifilters festgelegt. Es folgt anschliessend der
        % Schwellwert der linearen Exzentrizitaet, ab der das Programm zu
        % arbeiten beginnt. Das Programm untersucht alle Konturlinienen,
        % das geglaetetten Spektrums, die in DefaultContourLevels angegeben
        % sind und verwendet die hoechste gefundene Exzentrizitaet.
        DefaultSpectrumSmoother = struct('Sigma', 16, 'Size', 128, 'Filter', []);
        DefaultLowPass = struct('RidgeLine', struct('SigmaInterval', 8:-1:4),↙
'AreaOfActivity', struct('Sigma', 6), 'Filter', []);
        DefaultEccentricityThreshold = 0.6;
        DefaultContourLevels = 0.55:0.05:0.75;

    end



    properties(Access = private)

        % Es folgen die Definitionen der internen Variablen des Programs.
        % Die Variablen werden durch den Konstruktor oder ander Methoden
        % gesetzt.
        WorkingImage = [];
        FourierPaddingSize = 0;
        FourierSpectra = struct('Complex', [], 'LogarithmicAbsolut', [], 'Absolut', [],↙
'SmoothLogarithmicAbsolut', [], 'NormedSmoothLogarithmicAbsolut', [], 'SmoothAbsolut',↙
[], 'ComplexFiltered', []);
        EccentricityAndAngleData = [];
        OrientationAngle = 0;
        ContourLevelHeightFraction = 0;

    end



    properties(Access = public)

        % Diese Variablen koennen gesetzt werden. Am Anfang sind sie auf
        % die Standardwerte gestellt.
        SpectrumSmoother = DEGRIND.DefaultSpectrumSmoother;
        LowPass = DEGRIND.DefaultLowPass;
        ContourLevels = DEGRIND.DefaultContourLevels;
        EccentricityThreshold = DEGRIND.DefaultEccentricityThreshold;
```

```
    end



    methods

        function this = DEGRIND(Image, Height)

            % Das Bild muss ein Graustufenbild sein. Ein Bild mit drei
            % Farbkannaelen wird nicht akzeptiert. Das Bild muss mindestens
            % 24 mal 24 Groesse haben.
            assert(ndims(Image) == 2, 'DEGRIND::DEGRIND: Das Bild (Argument 1) muss ein↙
Graustufenbild sein.');
            assert(all(size(Image) > [23 23]), 'DEGRIND::DEGRIND: Das Bild (Argument 1)↙
muss mindestens eine Groesse von 24x24 haben.');

            % Die angegebene Hoehe darf die maximale Hohe nicht
            % ueberschreiten und darf nicht kleiner als 24 sein.
            assert(isscalar(Height), 'DEGRIND::DEGRIND: Die Hoehe (Argument 2) muss ein↙
Skalar sein.');
            assert((round(Height) == Height) && (Height < (DEGRIND.MaximumHeight + 1))↙
&& (Height > 23), sprintf('DEGRIND::DEGRIND: Die Bildhoehe muss ganzzahlig, mindestens↙
24 und hoechstens %d sein.', DEGRIND.MaximumHeight));

            % Das Bild wird verkleinert und es wird geprueft, ob die maximal
            % zulaessige Breite nicht ueberschritten wird.
            Image = imresize(Image, [Height NaN], 'bicubic');
            assert((size(Image, 2) < (DEGRIND.MaximumWidth + 1)) && (size(Image, 2) >↙
23), sprintf('DEGRIND::DEGRIND: Die Bildbreite darf %d nicht ueberschreiten und nicht↙
kleiner als 24 sein. Sie ist %d.', DEGRIND.MaximumWidth, size(Image, 2)));

            % Das Bild wird in der Klasseneigenschaft abgespeichert. Die
            % Groesse des Spektrums ist die groesse der langsten Bildseite.
            % Anschliessend werden die Speicherplaetze der Spektren
            % vorfreigegeben. Die Spaktrum Box sollte immere eine
            % Zweierpotenz sein und die Zahlen sollten nicht relativ prim
            % sein.
            this.WorkingImage = Image;
            PowersOfTwo = 2.^[5 6 7 8 9 10];
            FourierPaddingSizeIndex = find(PowersOfTwo >= max(size(this.WorkingImage)),↙
1);
            this.FourierPaddingSize = PowersOfTwo(1, FourierPaddingSizeIndex);
            this.FourierSpectra.Complex = zeros(this.FourierPaddingSize);
            this.FourierSpectra.LogarithmicAbsolut = zeros(this.FourierPaddingSize);
            this.FourierSpectra.Absolut = zeros(this.FourierPaddingSize);
            this.FourierSpectra.SmoothLogarithmicAbsolut = zeros(this.↙
FourierPaddingSize);
            this.FourierSpectra.SmoothAbsolut = zeros(this.FourierPaddingSize);

        end

    end
```

```
    methods(Access = public)

        % Laesst das Programm ablaufen. Das Programm entscheidet, selbst,
        % ob ein Schliff vorliegt oder nicht. Falls die lineare
        % Exzentrizitaet der Hoehenlinien nicht hoch genug ist, wird nicht
        % gefiltert.
        function DegrindedImage = Run(this)

            % Zunaechst werden alle Fourierspektren berechnet und dann in
            % der Klasse fuer die nachfolgenden Funktion erreichbar
            % abgespeichert.
            this.CalculateFourierSpectra();

            % Falls die Hoehenlinien eine lineare Exzentrizitaet hoeher als
            % der Schwellwert haben, dann ist Success true.
            Success = this.ProcessContourLevels();
            if(~Success)
                DegrindedImage = this.WorkingImage;
                return;
            end

            % Nachfolgend wird der entsprechende Low Pass Filter erzeugt.
            % Die Funktion liest und schreibt Klasseneigenschaften.
            this.CreateLowPass();

            % Nun wird der Low Pass auf das komplexe Fourier Spektrum
            % angewandt. Die Funktion bearbeitet Klassenvariablen.
            this.ApplyLowPassFilter();

            % Der Rueckgabewert ist das ruecktransformierte Bild.
            DegrindedImage = this.InverseTransformation();

        end

    end



    methods(Access = private)

        % Diese Funktion berechnet die verschiedenen benoetigten
        % Fourierspektren. Es wird die Matlab Fast Fourier Transformation
        % verwendet, diese Funktion liefert normatlerweise komplexe zahlen.
        function [] = CalculateFourierSpectra(this)

            % Fourierspektrum berechnen und das Spektrum speichern. Das
            % Shiften des Spektrums verischert, das die Nullfrequenzen
            % zentriert sind.
            this.FourierSpectra.Complex = fft2(this.WorkingImage, this.↙
FourierPaddingSize, this.FourierPaddingSize);
            this.FourierSpectra.Complex = fftshift(this.FourierSpectra.Complex);

            % Die Absolutbetraege des Spektrums werden berechnet und
            % gespeichert. Des Weiteren wird eine logarithmische Gewichtung
            % der Werte durchgefuehrt und gespeichert.
```

```
            this.FourierSpectra.Absolut = abs(this.FourierSpectra.Complex);
            this.FourierSpectra.LogarithmicAbsolut = log(this.FourierSpectra.Absolut);

            % Der Spektrum Filter wird erstellt und die Spektren damit
            % geglaettet. Die Spektren werden widerum abgespeichert.
            this.SpectrumSmoother.Filter = fspecial('gaussian', [this.SpectrumSmoother.↙
Size this.SpectrumSmoother.Size], this.SpectrumSmoother.Sigma);
            this.FourierSpectra.SmoothLogarithmicAbsolut =  imfilter(this.↙
FourierSpectra.LogarithmicAbsolut, this.SpectrumSmoother.Filter, 'symmetric');
            this.FourierSpectra.SmoothAbsolut = imfilter(this.FourierSpectra.Absolut,↙
this.SpectrumSmoother.Filter, 'symmetric');

            % Das normierte Spektrum wird noch berechnet. Dieses Spektrum
            % wird zur ermittlung der Hoehenlinien verwendet.
            Maximum = max(max(this.FourierSpectra.SmoothLogarithmicAbsolut));
            Minimum = min(min(this.FourierSpectra.SmoothLogarithmicAbsolut));
            this.FourierSpectra.NormedSmoothLogarithmicAbsolut = (this.FourierSpectra.↙
SmoothLogarithmicAbsolut - Minimum)/(Maximum - Minimum);

        end



        % Es wird nun in den Hoehenlinien nach der hoechsten linearen
        % Exzentrizitaet der umschriebenen Ellipse gesucht.
        function [Success] = ProcessContourLevels(this)

            % Success ist auf false voreingestellt.
            Success = false;

            % Alle Contour Levels werden durchsucht, falls keine
            % Hoehenlinie in ihrer Exzentrizitaet den Threshold uebersteigt,
            % dann wird nichts gemacht. Success ist nur dann true, falls
            % eine solche Hoehenlinie gefunden wird.
            ContourMask = false(this.FourierPaddingSize);
            n = numel(this.ContourLevels);
            this.EccentricityAndAngleData = zeros(3, n);
            for i = 1:1:n

                fprintf('DEGRIND::ProcessContourLevels: untersuche Hoehenlinie bei %f↙
... \n', this.ContourLevels(1, i));

                % Die Punkte der entsprechenden Hoehenlinie auslesen
                % lassen. Die Punkte werden sodann in lineare Indexe
                % verwendelt. Mit diesen Indexen kann die Hoehenlinie in
                % eine Maske gezeichnet werden.
                ContourLineData = contourc(this.FourierSpectra.↙
NormedSmoothLogarithmicAbsolut, [this.ContourLevels(1, i) this.ContourLevels(1, i)]);
                ContourRowIndexes = ContourLineData(2, 2:end);
                ContourColumnIndexes = ContourLineData(1, 2:end);
                ContourRowIndexes = ceil(ContourRowIndexes);
                ContourColumnIndexes = ceil(ContourColumnIndexes);
                ContourLinearIndexes = ContourRowIndexes + this.FourierPaddingSize .*↙
(ContourColumnIndexes - 1);
                ContourMask(:, :) = false;
```

```
                ContourMask(ContourLinearIndexes) = true;

                % Die Anzahl der weissen Objekte in der Hoehenlinienmaske
                % wird nun ermittlet. Wenn es sich nicht um eine einziges Obbjekt
                % handelt, dann wurde die Linie abgschnitten. Die Werte
                % koennen dann nicht ermittelt werden.
                cc = bwconncomp(ContourMask, 8);
                if(cc.NumObjects ~= 1)
                    fprintf('DEGRIND::ProcessContourLevels: Hoehenlinie bei %f ist↙
nicht geschlossen, Daten werden nicht verwendet ...\n', this.ContourLevels(1, i));
                    this.EccentricityAndAngleData(1, i) = NaN;
                    this.EccentricityAndAngleData(2, i) = NaN;
                    this.EccentricityAndAngleData(3, i) = this.ContourLevels(1, i);
                else
                    ContourOrientation = regionprops(ContourMask, 'orientation');
                    ContourEccentricity = regionprops(ContourMask, 'eccentricity');
                    fprintf('DEGRIND::ProcessContourLevels: Exzentrizitaet: %f↙
Orienterung: %f ... \n', ContourEccentricity.Eccentricity, ContourOrientation.↙
Orientation);
                    this.EccentricityAndAngleData(1, i) = ContourEccentricity.↙
Eccentricity;
                    this.EccentricityAndAngleData(2, i) = ContourOrientation.↙
Orientation;
                    this.EccentricityAndAngleData(3, i) = this.ContourLevels(1, i);
                end

            end

            % Es werden nun alle Exzentrizitaetswert, die ueber dem
            % Schwellwert liegen herangezogen. Falls es keine solchen
            % gibt, dann wird abgebrochen.
            [~, OverThresholdColumnIndexes] = find(this.EccentricityAndAngleData(1, :)↙
> this.EccentricityThreshold);
            EccentricityAndAngleDataOverThreshold = this.EccentricityAndAngleData(:,↙
OverThresholdColumnIndexes);
            if(size(EccentricityAndAngleDataOverThreshold, 2) == 0)
                fprintf('DEGRIND::ProcessContourLevels: Die lineare Exzentrizitaet↙
uebersteigt den Schwellwert fuer keine Hoehe, das Bild hat keinen Schliff, nichts tun↙
...\n');
                return;
            end

            % Falls es werte ueber dem Threshold gibt, dann wird das
            % Maximum berechnet und der Winkel an dieser Stelle
            % verwendet. Die ContourLevelHeightFraction gibt an in welcher
            % relativen Hoehe sich die maximale lineare Exzentrizitaet
            % befindet. Je Hoeher dieser Wert desto schmaler wird die
            % RidgeLine in CreateLowPass.
            [~, ColumnIndex] = max(EccentricityAndAngleDataOverThreshold(1, :));
            this.OrientationAngle = EccentricityAndAngleDataOverThreshold(2,↙
ColumnIndex);
            this.ContourLevelHeightFraction = (EccentricityAndAngleDataOverThreshold(3,↙
ColumnIndex) - this.ContourLevels(1, 1))/(this.ContourLevels(1, end) - this.↙
ContourLevels(1, 1));
```

```
            fprintf('DEGRIND::ProcessContourLevels: Die Ausrichtung des Spektrums liegt bei %f gegen die Horizontale ...\n',  this.OrientationAngle);
            fprintf('DEGRIND::ProcessContourLevels: Die hoechste lineare Exzentrizitaet liegt bei %f Anteilen des Hoehenlinienintervalls ...\n', this.ContourLevelHeightFraction);

            Success = true;

        end



        % Diese Funktion erzeugt eine an den Schliffverlauf angepassten Low
        % Pass Filter. Die Dimensionierung der Ridge Line variiert je
        % nachdem an welcher Hoehnlinie die groesste Exzentrizitaet
        % auftritt. Die Funktion arbeitet an der Variable this.LowPass.
        function [] = CreateLowPass(this)

            % Falls der Winkel 90 oder -90 ist dann sind BuildFilterSize und
            % FilterSize gleich gross.
            FilterSize = this.FourierPaddingSize;
            BuildFilterSize = ceil(FilterSize * (sind(abs(this.OrientationAngle)) + cosd(abs(this.OrientationAngle))));
            Offset = ceil(FilterSize * (sind(abs(this.OrientationAngle)) * cosd(abs(this.OrientationAngle))));

            % Die RidgeLine wird erzeugt. Der Sigmawert wird dabei
            % umgekehrt proportinal zu this.ContourLevelHeightFraction
            % gewaehlt.
            RidgeLineSigmaIndex = 1 + round((numel(this.LowPass.RidgeLine.SigmaInterval) - 1) * this.ContourLevelHeightFraction);
            RidgeLineSigma = this.LowPass.RidgeLine.SigmaInterval(RidgeLineSigmaIndex);
            RidgeLine = fspecial('gaussian', [1, BuildFilterSize], RidgeLineSigma);
            RidgeLine = max(max(RidgeLine)) - RidgeLine;

            fprintf('DEGRIND::CreateLowPass: RidgeLine Sigma is %f ...\n', RidgeLineSigma)

            % Die AreaOfActivity wird nun erzeugt.
            AreaOfActivity = fspecial('gaussian', [BuildFilterSize, 1], this.LowPass.AreaOfActivity.Sigma);

            % Der Filter ist das Aeussereprodukt zwischen RidgeLine und
            % AreaOfActivity. Der BuildFilter ist etwas groesser als das
            % Fourier Spektrum und der enduelltige Filter. Somit wird die
            % Rotation des Filters ohne Verlust ermoeglicht.
            BuildFilter = AreaOfActivity * RidgeLine;

            % Wenn die BuildFilterSize und die FilterSize nicht gleich
            % gross ist, dann muss der Filter rotiert und anschliessend
            % ausgeschnitten werden.
            if(BuildFilterSize ~= FilterSize)
                LowPassFilterContainer = imrotate(BuildFilter, this.OrientationAngle, 'bilinear', 'loose');
                LowPassFilter = LowPassFilterContainer(Offset:(Offset + FilterSize -
```

```
1), Offset:(Offset + FilterSize - 1));
            end

            % Der Filter wird nachfolgend invertiert und auf das Intervall
            % 0 bis 1 normiert und anschliessend gespeichert.
            LowPassFilter = max(max(LowPassFilter)) - LowPassFilter;
            LowPassFilter = (LowPassFilter - min(min(LowPassFilter)))/(max(max↙
(LowPassFilter)) - min(min(LowPassFilter)));
            this.LowPass.Filter = LowPassFilter;

        end


        % Diese Methode wendet den erzeugten Low Pass Filter auf das
        % komplexe Fourier Spektrum an und speichert es in
        % this.FourierSpectra.ComplexFiltered.
        function [] = ApplyLowPassFilter(this)

            % Die Filterung des komplexen Fouriersoektrums ist eine
            % elementweise Multiplikation mit dem Filter im Frequenzraum.
            this.FourierSpectra.ComplexFiltered = this.FourierSpectra.Complex .* this.↙
LowPass.Filter;

        end



        % Nach der Filterung des Spektrums wird wird es von dieser Funktion
        % zuruecktransformiert und wider auf Ausgangsbildgroesse gebracht.
        function [ResultImage] = InverseTransformation(this)

            % Die Invertierung der Fourier Transformation leifert komplexe
            % Zahlen. Um wider ein Graustufenbild zu erhalten muss der
            % Absolutbetrag ermittelt und mat2gray angewandt werden.
            ResultImage = mat2gray(abs(ifft2(this.FourierSpectra.ComplexFiltered)));

            % Aus dem ruecktransfomrierten Bild muss noch der passende
            % Bereich ausgeschnitten werden.
            ResultImage = ResultImage(1:size(this.WorkingImage, 1), 1:size(this.↙
WorkingImage, 2));

        end

    end

end
```

```
classdef DEILLUM < hgsetget
    %DEILLUM DEILLUMINATION PROGRAM
    %   deillumination program

    properties(Constant = true)

        % Die nachfolgend festgelegten Hoechstwerte fuer die Seiten des
        % Bildes duerfen nicht ueberschritten werden. Die Werte sind
        % festgelgt um eine Ueberforderung zu vermeiden.
        MaximumHeight = 640;
        MaximumWidth = 640;

        % Es folgen die Standardwerte, die das Programm verwendet, falls
        % sie nicht anders gesetzt werden.
        DefaultIlluminationSmoother = struct('Sigma', 12, 'Size', 64, 'Filter',  []);
        DefaultHeight = 512;
        DefaultBlockSize = 16;
        DefaultDilationMethodStructuringElementSize = 32;

        % Die Methode ist entweder smooth oder morph
        DefaultMethod = 'morph';

    end



    properties(Access = public)

        % Alle Eigenschaften auf Standardwerte setzen.
        Height = DEILLUM.DefaultHeight;
        IlluminationSmoother = DEILLUM.DefaultIlluminationSmoother;
        BlockSize = DEILLUM.DefaultBlockSize;
        DilationMethodStructuringElementSize = DEILLUM.↙
DefaultDilationMethodStructuringElementSize;
        WorkingImage = [];
        Method = DEILLUM.DefaultMethod;

    end



    methods

        function this = DEILLUM(Image, Height, Method)

            % Das Bild muss ein Graustufenbild sein. Ein Bild mit drei
            % Farbkannaelen wird nicht akzeptiert. Das Bild muss mindestens
            % 24 mal 24 Groesse haben.
            assert(ndims(Image) == 2, 'DEILLUM::DEILLUM: Das Bild (Argument 1) muss ein↙
Graustufenbild sein.');
            assert(all(size(Image) > [23 23]), 'DEILLUM::DEILLUM: Das Bild (Argument 1)↙
muss mindestens eine Groesse von 24x24 haben.');

            % Die angegebene Hoehe darf die maximale Hohe nicht
            % ueberschreiten und darf nicht kleiner als 24 sein.
```

```
            assert(isscalar(Height), 'DEILLUM::DEILLUM: Die Bildhoehe (Argument 1) muss
ein Skalar sein sein.');
            assert((round(Height) == Height) && (Height < (DEILLUM.MaximumHeight + 1))
&& (Height > 23), sprintf('DEILLUM::DEILLUM: Die Bildhoehe muss ganzzahlig, mindestens
24 und hoechstens %d sein.', DEILLUM.MaximumHeight));

            % Das Bild wird verkleinert und es wird geprueft, ob die maximal
            % zulaessige Breite nicht ueberschritten wird.
            Image = imresize(Image, [Height NaN], 'bicubic');
            assert((size(Image, 2) < (DEILLUM.MaximumWidth + 1)) && (size(Image, 2) >
23), sprintf('DEILLUM::DEILLUM: Die Bildbreite darf %d nicht ueberschreiten und nicht
kleiner als 24 sein. Sie ist %d.', DEILLUM.MaximumWidth, size(Image, 2)));

            % Es wird nun ueberprueft, ob eine korrekte Methode angegeben
            % wurde.
            assert(ischar(Method), 'flascher Datentyp (Arguemnt 3)');
            assert(strcmp(Method, 'smooth') || strcmp(Method, 'morph'), 'DEILLUM::
DEILLUM: unbekannte Methode (Argument 3)');
            this.Method = Method;
            % Wenn alles in Ordnung ist, wird das Bild abgespeichert.
            this.WorkingImage = Image;

        end

    end



    methods(Access = public)

        function DeilluminatedImage = Run(this)

            switch this.Method
                case 'smooth'
                    this.DeilluminateImage();
                case 'morph'
                    this.DeilluminateImageDilation();
            end
            DeilluminatedImage = this.WorkingImage;

        end

    end



    methods(Access = private)

        function [] = DeilluminateImage(this)

            % Zunaechst wird herausgefunden, ob das Bild eher dunkel oder
            % hell ist. Dann wird entsprechend IsDark auf richtig oder
            % falsch gestellt. Je nachdem ob das Bild dunkel oder hell ist,
            % werden die dunklen Werte erhoeht oder die hellen Werte
            % erniedrigt.
```

```
            NumberAbove127 = sum(sum(this.WorkingImage > 127));
            FractionAbove127 = NumberAbove127/numel(this.WorkingImage);
            if(FractionAbove127 > 0.5)
                fprintf('DEILLUM::DeilluminateImage: image is more likely bright
(anteil: %f) ...\n', FractionAbove127);
                IsDark = false;
            else
                fprintf('DEILLUM::DeilluminateImage: image is more likely dark (anteil:
%f) ...\n', FractionAbove127);
                IsDark = true;
            end

            % Der Helligkeitsverlauf wird mittels Blockprozessing
            % ermittelt. Das Blockimage wird sodann mit einem Gaussfilter
            % geglaettet.
            BlockFunction = @(block_struct) mean2(block_struct.data);
            BlockImage = blockproc(this.WorkingImage, [this.BlockSize, this.BlockSize],
BlockFunction, 'PadMethod', 'symmetric');
            this.IlluminationSmoother.Filter = fspecial('gaussian', [this.
IlluminationSmoother.Size this.IlluminationSmoother.Size], this.IlluminationSmoother.
Sigma);
            BlockImage = imfilter(BlockImage, this.IlluminationSmoother.Filter,
'symmetric');

            % Wenn das Bild dunkel ist, dann werden Werte addiert. Es
            % werden die Differenzwerte zum Maximum addiert. Wenn das Bild
            % Hell ist, werden die Differenzwerte zum Minimum subtrahiert.
            ChangeGrayValues = imresize(BlockImage, size(this.WorkingImage),
'nearest');
            ChangeGrayValues = round(ChangeGrayValues);
            if(IsDark)
                ChangeGrayValues = max(max(ChangeGrayValues)) - ChangeGrayValues;
            else
                ChangeGrayValues = min(min(ChangeGrayValues)) - ChangeGrayValues;
            end
            this.WorkingImage = double(this.WorkingImage) + ChangeGrayValues;
            if(IsDark)
                Above255Mask = this.WorkingImage > 255;
                this.WorkingImage(Above255Mask) = 255;
            else
                Below0Mask = this.WorkingImage < 0;
                this.WorkingImage(Below0Mask) = 0;
            end

        end



        function [] = DeilluminateImageDilation(this)

            % Zunaechst wird herausgefunden, ob das Bild eher dunkel oder
            % hell ist. Dann wird entsprechend IsDark auf richtig oder
            % falsch gestellt. Je nachdem ob das Bild dunkel oder hell ist,
            % werden die dunklen Werte erhoeht oder die hellen Werte
            % erniedrigt.
```

```
            %NumberAbove127 = sum(sum(this.WorkingImage > 127));
            %FractionAbove127 = NumberAbove127/numel(this.WorkingImage);
            %f(FractionAbove127 > 0.5)
            %    fprintf('DEILLUM::DeilluminateImage: image is more likely bright
(anteil: %f) ...\n', FractionAbove127);
            %    IsDark = false;
            %else
            %    fprintf('DEILLUM::DeilluminateImage: image is more likely dark
(anteil: %f) ...\n', FractionAbove127);
            %    IsDark = true;
            %end

            % Der Hintergrundhelligkeitsverlauf wird mit morphologischer
            % Oeffnung ermittelt.
            BackgroundIllumination = imopen(this.WorkingImage, strel('disk', this.
DilationMethodStructuringElementSize));

            %[x, y] = meshgrid(1:1:size(BackgroundIllumination, 2), 1:1:size
(BackgroundIllumination, 1));
            %surf(x, y, double(BackgroundIllumination)); colormap(jet); colorbar;
            %waitforbuttonpress;


            % Wenn das Bild dunkel ist, dann werden Werte addiert. Es
            % werden die Differenzwerte zum Maximum addiert. Wenn das Bild
            % Hell ist, werden die Differenzwerte zum Minimum subtrahiert.
            %if(IsDark)
            %    ChangeGrayValues = max(max(BackgroundIllumination)) -
BackgroundIllumination;
            %else
            %    ChangeGrayValues = min(min(BackgroundIllumination)) -
BackgroundIllumination;
            %end
            %this.WorkingImage = this.WorkingImage + ChangeGrayValues);
            %if(IsDark)
            %    Above255Mask = this.WorkingImage > 255;
            %    this.WorkingImage(Above255Mask) = 255;
            %else
            %    Below0Mask = this.WorkingImage < 0;
            %    this.WorkingImage(Below0Mask) = 0;
            %end

            this.WorkingImage = this.WorkingImage - BackgroundIllumination;
            %this.WorkingImage = imadjust(this.WorkingImage);

            %imshow(this.WorkingImage);
            %waitforbuttonpress;

        end


    end


end
```