Institut für Elektromechanische Konstruktionen
Technische Universität Darmstadt

Prof. Dr.-Ing. H.F. Schlaak

1967

Bachelorarbeit
für
Herrn Dimitri Haas

**Oberflächenbestimmung von Nanodrähten durch Bildverarbeitung**

Am Institut für Elektromechanische Konstruktionen wird an der Entwicklung einer neuartigen, auf eindimensionalen Nanostrukturen basierenden Mikrobatterie geforscht. Durch die Verwendung von Nanodrähten können sehr hohe Oberflächen-zu-Volumenverhältnisse erreicht werden, wodurch sich makroskopische Effekte verstärken lassen. Die durch metallische Nanodrähte hervorgerufene Oberfläche kann bisher nur durch aufwändige chemische Verfahren bestimmt werden (z.B. BET-Methode). Im Rahmen dieser Arbeit soll daher eine einfache Möglichkeit zur Abschätzung der tatsächlichen Oberfläche von großflächig aufgebrachten Nanodrähten entstehen.

Durch das Festlegen von Anforderungen an Auflösung, Vergrößerung, Helligkeit und Kontrast der mit dem Raster-Elektronen-Mikroskop aufgenommenen Bilder sollen zuverlässig auswertbare Bilddaten von Nanodrähten entstehen.

Diese Bilder sollen anschließend mit Methoden der Bildverarbeitung und bei Bedarf des Maschinellen Lernens untersucht werden. Ziel ist es, eine Aussage über die tatsächliche Oberflächenvergrößerung durch die zufällig verteilten Nanodrähte zu treffen.

Im letzten Schritt sollen die so erhaltenen Messergebnisse mit einer herkömmlichen chemischen Messung verglichen und validiert werden, um eine Aussage über die Genauigkeit des entwickelten Verfahrens zu erhalten.

Darmstadt, den 11.06.2018

H. Schlaak

Beginn der Arbeit: 11.06.2018
Ende der Arbeit: 12.11.2018
Seminar: 29.11.2018

Betreuer: Konja Wick, M.Sc.

# Oberflächenbestimmung von Nanodrähten durch Bildverarbeitung

**von Dimitri Haas**
Bachelorarbeit, vorgelegt im November 2018
Betreuer: Konja Wick, M.Sc.
Institut für Elektromechanische Konstruktionen

TECHNISCHE
UNIVERSITÄT
DARMSTADT

## Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Dimitri Haas, die vorliegende Bachelorarbeit gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden. Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein. Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

---

**English translation for information purposes only:**
**Thesis Statement pursuant to § 22 paragraph 7 and § 23 paragraph 7 of APB TU Darmstadt**
I herewith formally declare that I, Dimitri Haas, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form. I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once. In the submitted thesis the written copies and the electronic version for archiving are pursuant to § 23 paragraph 7 of APB identical in content. For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

Darmstadt, den 12. November 2018

(Dimitri Haas)

## Zusammenfassung

Ziel dieser Arbeit ist die Entwicklung einer Methode zur Abschätzung der Oberflächenvergrößerung durch großflächig aufgebrachte Nanodrähte anhand auswertbarer Bilddaten eines Rasterelektronenmikroskops. Untersucht wurden dabei die Poren von sogenannten Templatfolien, welche zur galvanischen Erzeugung von Mikro- und Nanodrähten benötigt werden. Durch ausreichende Vergrößerung der auflaminierten Folie konnten Position und Verlauf der Poren mittels Techniken der Bildverarbeitung erfasst und ein Rückschluss auf die Mantelfläche der anschließend erzeugten Drähte gezogen werden.

Mithilfe der Software MATLAB und des dazugehörigen Plug-Ins *Image Processing Toolbox* wurden im ersten Schritt die Aufnahmen durch Zuschnitt und Skalierung für die weitere Verarbeitung adaptiert. Darauf folgte die Untersuchung und Anwendung zweier Methoden der Informationsgewinnung im Rahmen der Kanten- und Kreisdetektion.
Erstere projiziert mittels Hough-Transformation gefundene Kreise auf die Aufenthaltsorte der Poren und berechnet in einem geometrischen Kontext die gesamte Umfangslinie. Erschwerend kam hinzu, dass diese Poren nicht ausschließlich isoliert auf der Oberfläche vorliegen, sondern sich beliebig überschneiden können. Auf den daraus entstehenden trigonometrischen Rechenaufwand wird detailliert eingegangen.
Mit dem Ziel die Berechnung zu beschleunigen, wurde als zweite Möglichkeit die durch den Canny Algorithmus extrahierte Kontur und deren Pixelanzahl als Annäherung für Länge und Aufenthaltsort der Umfangslinie vorgestellt. Die resultierenden Messabweichungen werden mithilfe von Streugrößen auf Systematik und die Möglichkeit einer Kompensation anhand vieler Fallbeispiele untersucht.
Die aus beiden Methoden gewonnene Umfangslinienlänge wird in einem letzten Schritt zur Bestimmung der Mantelfläche und der damit verbundenen Oberflächenvergrößerung genutzt.

In einer Gegenüberstellung wurden anschließend Zeitperformance, Robustheit bezüglich unterschiedlicher Aufnahmeparameter und die geschätzte Mantelfläche beider Methoden miteinander verglichen. Hierbei überzeugte vor allem die geometrische Methode mit einer präzisen Kreisdetektion und -lokalisation, wodurch eine genaue Berechnung der Oberflächenvergrößerung möglich war. Negativ hat sich zum Teil nur die Ausführungszeit von bis zu zwei Minuten ausgewirkt, wenn der für den Algorithmus benötigte Sensitivitätswert noch generisch berechnet werden musste. Aufgrund der genauen Ergebnisse der geometrischen Berechnung, wurden diese als Referenzwert für die Beurteilung der pixelbasierten Methode verwendet. Schlussfolgerend konnte festgestellt werden, dass die pixelbasierte Methode um den Faktor 20-30 schneller arbeitet als ihr geometrischer Pendant – sich jedoch nicht besonders robust gegenüber unterschiedlichen Bildparametern zeigt. Die Schwächen liegen insbesondere in niedrigen Auflösungen und kleinen Abbilungsmaßstäben der REM-Aufnahmen. Messunsicherheiten sind deshalb nicht ohne weiteres vernachlässigbar.

Die Arbeit schließt mit einem Ausblick auf weitere, mögliche Ansatzpunkte zur Erhöhung der Genauigkeit und Performance der Oberflächenbestimmung.

## Inhaltsverzeichnis

# 1 Einleitung

Am Institut für Elektromechanische Konstruktionen der TU Darmstadt wird an der Entwicklung einer neuartigen, auf eindimensionalen Nanostrukturen basierenden Mikrobatterie geforscht. Durch die Verwendung von Nanodrähten können sehr hohe Oberflächen-zu-Volumenverhältnisse erreicht werden, wodurch sich makroskopische Effekte verstärken lassen. Die durch metallische Nanodrähte hervorgerufene Oberfläche kann bisher nur durch aufwändige chemische Verfahren bestimmt werden (z.B. BET-Methode). Im Rahmen dieser Arbeit soll daher eine einfache Möglichkeit zur Abschätzung der tatsächlichen Oberfläche von großflächig aufgebrachten Nanodrähten entstehen.

Durch das Festlegen von Anforderungen an Auflösung, Vergrößerung, Helligkeit und Kontrast der mit dem Raster-Elektronen-Mikroskop aufgenommenen Bilder, sollen zuverlässig auswertbare Bilddaten von Nanodrähten generiert werden können.

Diese Bilder sollen anschließend mit Methoden der Bildverarbeitung untersucht werden. Ziel ist es, eine Aussage über die tatsächliche Oberflächenvergrößerung der zufällig verteilten Nanodrähte zu treffen.

## 1.1 Motivation

Die Bestimmung der aktiven Oberfläche eines Nanodraht-Arrays ist in der Regel mit einem chemischen Messverfahren [1] verbunden, das in einem Labor durchgeführt werden muss. Damit geht eine terminliche und personelle Vorbereitung sowie ein filigraner Versuchssaufbau einher, der die Berücksichtigung von Messunsicherheiten und den Umstand mehrerer Anläufe nötig macht. Dies resultiert in einem hohen Zeit- und Kostenfaktor für den Ingenieur.
Aus diesem Grund ist es besonders interessant im Rahmen dieser Arbeit ein Verfahren zu entwickeln, mit dessen Hilfe sich die Abschätzung der Array-Oberfläche deutlich beschleunigen lässt.

Die Anfertigung von Aufnahmen mit einem Rasterelektronenmikroskop lassen sich vergleichsweise schnell anfertigen. Sind die gewünschten Aufnahmeparameter eingestellt, können viele Bilder in kurzer Zeit aufgenommen und gespeichert werden. Diese Bilder können von da an automatisiert vom Rechner mithilfe verschiedenster Techniken aus der Disziplin *Computer Vision* gesichtet, analysiert und ausgewertet werden. Eine abschätzende Aussage zur Oberflächenvergrößerung kann auf diese Weise innerhalb weniger Sekunden bis wenigen Minuten getroffen und zur weiteren Beurteilung herangezogen werden. Eine einzige Oberflächenmessung im Labor kann dagegen nicht selten bis zu einer Stunde dauern.

## 1.2 Gliederung der Arbeit

Die Arbeitsschritte zur Extraktion von Informationen aus einem Bild im Rahmen der Bildverarbeitung umfassen in dieser Arbeit die Aufbereitung des Bildes in Kapitel 2, die Extraktion

[1] Die Oberfläche lässt sich beispielsweise mithilfe der *Cyclovoltammetrie* [6] abschätzen. Dies ist ein analytisches Verfahren, mit dessen Hilfe man Elektrodenprozesse beobachtet und Ausschläge in der Widerstandskennlinie in direkten Zusammenhang mit der chemisch aktiven Oberfläche setzt.

der Kanten in Kapitel 3.3, die Suche nach geometrischen Kreisen in Kapitel 3 und die anschließende Berechnung der geometrischen Oberfläche der zu untersuchenden Struktur in Kapitel 4.1.

Dabei werden zwei Ansätze der Oberflächenerfassung verfolgt:

1. **Kreisdetektion mithilfe von Hough-Transformation und der MATLAB-Funktion imfindcircles**
   Wenn mehr Rechenzeit zur Verfügung steht, soll eine genauere Variante mit Hilfe der Hough-Transformation [1] durchgeführt werden. Dabei wird das Bild nach kreisähnlichen Strukturen abgesucht und bei jedem Treffer ein idealer Kreis an der jeweiligen Position angenommen. Da sich diese auch überschneiden und damit kleinere Mantelflächen zur Folge haben können, müssen die genauen Positionen der Kreise zwischengespeichert und geometrisch untersucht werden. Dies wird in Abschnitt 4.1 näher beleuchtet.

2. **Analyse der detektierten Kanten**
   In dieser Methode wird das Bild auf seine Kanten und Konturen untersucht, um diese anschließend als Grundlage für eine Oberflächenberechnung heranzuziehen. Als Kantenfilter wird der sogenannte Canny-Filter [2] näher untersucht, da dieser heute als der profilierteste Kantenfilter gilt. Anhand der Anzahl der detektierten Pixel wird die Länge des Umrisses unter Berücksichtigung des Diskritisierungsfehlers durch Kreisrasterung abgeschätzt. Dieser Umriss kann in einem weiteren Schritt zur Berechnung der Mantelfläche herangezogen werden. Dieser Ansatz wird ab Abschnitt 4.2.1 behandelt.

Nach Vorstellung beider Abschätzungsverfahren werden diese in Kapitel 5 auf ihre Performance, Zuverlässigkeit und Fehlerkorrelation untersucht.

## 1.3 Werkzeuge

Für die Aufgabenstellung eignet sich eine Vielzahl von Werkzeugen und Programmiersprachen. Gut dokumentierte Frameworks zur Bildverarbeitung gibt es beispielsweise für Python (*OpenCV* [7]) und MATLAB (*Image Processing Toolbox* [8]).

In dieser Arbeit wird das Problem in erster Linie mit MATLAB in der Version `R2017b` gelöst. Dabei werden robuste, vorimplementierte Funktionen wie beispielsweise `imfindcircles` genutzt, welche mit Hilfe der Hough-Transformation hervorragende Ergebnisse hinsichtlich der detektierten Kreise zeigen. Die generische Bestimmung möglichst idealer Parameter zur Steuerung dieser Funktionen bildet einen wesentlichen Teil dieser Arbeit.

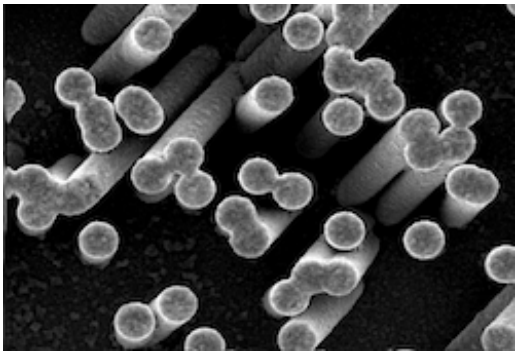
Weiter wird auch die Auswirkung unterschiedlicher Aufnahme- und Bildparameter auf das Ergebnis und die Performance der hier vorgestellten Berechnungsmethoden betrachtet.

## 1.4 Auswahl der zu untersuchenden Bildern

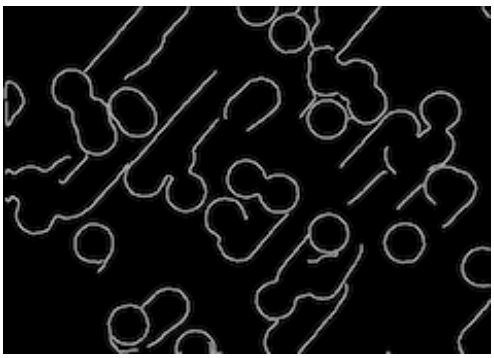
Vor Beginn der Untersuchung der Oberfläche stellt sich die Frage, ob die Analyse von vornerein durch eine gezielte Auswahl von Bildern positiv beeinflusst werden kann. So kann

sowohl eine Aufnahme der Nanodrähte (Abb. 1.1a) als auch die einer Templatfolie (Abb. 1.2a) als Ausgangsmaterial für die Berechnung in Betracht gezogen werden. Vorangegangene Kantendetektionen haben gezeigt, dass die Aufnahmen von Drähten eine Oberflächenberechnung sehr erschweren. Das liegt vor allem daran, dass die Querschnittsflächen der Drähte, welche nicht orthogonal zum Betrachter zeigen, optisch mit einem großen Teil ihrer Mantelfläche verschmelzen (in Abb. 1.1b an vielen diagonal verlaufenden Bildern sichtbar) und damit eine Abgrenzung beider Flächenanteile sehr schwierig ist.

Aus diesem Grund werden vorrangig nur Aufnahmen von Templatfolien untersucht, da die Abgrenzungen der Poren auch nach der Kantendetektion weitestgehend erhalten bleiben (siehe Abb. 1.2b).

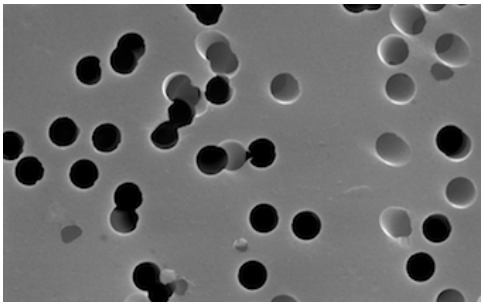


**(a)** Aufnahme mehrerer Nanodrähte mit einem Rasterelektronenmikroskop

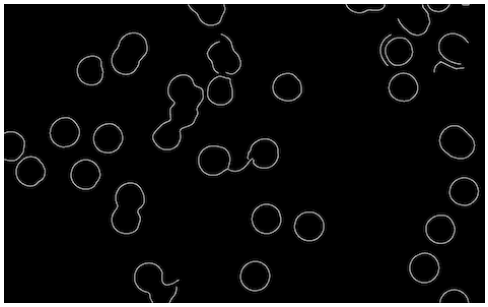


**(b)** Extraktion der Kanten mithilfe des Canny Algorithmus

**Abbildung 1.1.:** Anhand des Kantenbilds eines Nanodrahtarrays wird ersichtlich, dass Flächendiskontinuitäten nicht immer von einem Kantenfilter erfasst werden können.



**(a)** REM-Aufnahme einer auflaminierten Templatfolie mit gut sichtbaren Poren



**(b)** Auch hier fand eine Kantenextraktion durch den Canny Filter statt

**Abbildung 1.2.:** Verglichen zu Abb. 1.1a lassen sich Umfangslinien einzelner Löcher deutlich besser extrahieren, da ein ausreichender Kontrast zwischen den aufeinandertreffenden Flächen vorliegt.

# 2 Vorbereitung der REM-Aufnahmen

## 2.1 Bildzuschnitt

Bei den uns zur Verfügung stehenden Bilder handelt es sich um Aufnahmen aus einem Elektronenmikroskop. Diese werden im .tif-Format mit einer Auflösung von $1424 \times 968$ exportiert und gespeichert. Zusätzliche Meta-Informationen, wie beispielsweise Zoomfaktor, Auflösung, Helligkeit, Kontrast, angelegte Spannung und ein Maßbalken, werden mithilfe einer schwarzen Box am unteren, linken Rand des Bildes angezeigt. Diese Box überdeckt einen Teil des Bildes und muss deshalb vor der Untersuchung ausgeschnitten und bei Bedarf ausgelesen werden. Dieser Wegschnitt resultiert in einem etwas kleineren Bildausschnitt.

Für den in Abschnitt 2.3 berechneten Skalierungsfaktor sind die wichtigsten Parameter die Länge des Maßbalkens in Pixeln und der daneben aufgedruckte Wert in Mikrometern. Dieser Maßstab wird im betreffenden Abschnitt zur Berechnung des Skalierungsfaktors herangezogen. Dieser Faktor gibt an, wie viel die Länge eines Pixels in Mikrometern beträgt und ermöglicht damit erst die sinnvolle Einheitenkonvertierung.

Da die schwarze Infobox stets zum Gesamtbild die gleiche, relative Größe besitzt und an der gleichen, relativen Stelle lokalisiert ist, kann der Wegschnitt im Vorfeld festgelegt und mit einer einzelnen Zeile durchgeführt werden (s. Quelltext 2.1).

**Quelltext 2.1:** Wegschnitt der schwarzen Infobox

```
1 A = A(1:floor(end*0.85),:);
```

## 2.2 Anpassung der Auflösung

Um bereits eine im Laufe dieser Arbeit entstandene Erkenntnis vorzugreifen: Hoch aufgelöste Bilder können wesentlich besser auf Strukturen untersucht werden. Deshalb ist es wünschenswert die Auflösung gegebenenfalls anzupassen, wenn diese aufgrund von niedriger Auflösung bildverarbeitende Techniken erschweren.
In MATLAB können Bilder mit dem Befehl `imresize` beliebig in verschiedene Richtungen skaliert werden. In Quelltext 2.2 wird das Bild `Image` zunächst um 5 % in $x$-Richtung gestaucht und anschließend um den Faktor 2 aufgebläht.

**Quelltext 2.2:** Bildskalierung in MATLAB

```
1 [rows, columns] = size(Image);
2 scale = 2;
3 xScale = 0.95;
4 scaledImage = imresize(imresize(I,[rows columns*xScale]),scale);
```

Die in dieser Arbeit untersuchten REM-Bilder werden im Vorfeld um 5 % in $x$-Richtung gestaucht[1] und anschließend um den Faktor 1,4 skaliert, um Bildgrößen von etwa $1000 \times 2000$ zu erhalten[2].

## 2.3 Distanzmessung im Bild: Bestimmung des Skalierungsfaktors

Um herauszufinden, wie viele Mikrometer einem Pixel des zu analysierenden Bildes entsprechen, muss der Skalierungsfaktor $S$ bestimmt werden, welcher die Umrechnung von Pixeln in Mikrometern ermöglicht.

### Möglichkeit 1: Maßbalken ablesen

Oft sind (mikroskopischen) Bildern ein Maßbalken beigefügt, an dessen Länge man den Maßstab ablesen kann. Um diesen Balken auslesen zu können, kann beispielsweise mit *Template-Matching* [5] gearbeitet werden. Dazu wird eine Bilddatei eines weißen Balkens angelegt, damit diese als Template im Zielbild gesucht werden kan. Der Template-Matching-Algorithmus sucht das komplette Bild nach dem Brut-Force-Prinzip ab, bis das Template lokalisiert ist. Nach anschließender Pixelzählung des gefundenen Balkons, kann letztendlich der danebenstehende Zahlenwert mit den erfassten Pixeln in Relation gebracht und ein Skalierungsfaktor $S$ gebildet werden.

$$S := \frac{l_T}{l_M} \tag{2.1}$$

**Beispiel:** Beträgt die Länge $l_M$ des Maßbalkens 160 Pixel, welche laut nebenstehendem Zahlenwert $l_T$ einer Länge von $2\mu$m entspricht, so liegt der Skalierungsfaktor nach Gleichung 2.1 bei $\frac{2}{160}$. Wenn also nun eine Kantenlänge $l_{\text{px}}$ in Pixeln im Bild auf die echte Länge $l$ in Mikrometern untersucht werden soll, muss diese lediglich mit dem Skalierungsfaktor multipliziert werden:

$$l = l_{\text{px}} \cdot S \tag{2.2}$$

### Möglichkeit 2: Auslesen eines vorliegenden Datenblatts

Wenn die Kreise eine bereits bekannte Größe (durch Vorliegen eines Datenblatts) haben, kann dies bei der Lokalisierung im Bild als Information zur Bestimmung des Skalierungsfaktors genutzt werden.

[1] Damit wird die leichte Verzerrung des Bildes ausgeglichen, welche speziell bei diesem Rasterelektronenmikroskop auftraten.

[2] Aus mehreren Versuchsreihen wurde ersichtlich, dass eine Auflösung von etwa $1000 \times 2000$ ein guter Kompromis zwischen guter Detektion und Rechenaufwand darstellt.

**Beispiel: Bestimmung eines Skalierungsfaktor anhand Datenblatt**
Anhand des Templat-Datenblatts werden Löcher mit einem Radius $r_{\mathrm{D,px}}$ von 200 nm erwartet. Werden nach Durchlaufen eines Such-Algorithmus Kreise mit einem durchschnittlichen Radius von 27,6 Pixeln detektiert, so kann dieser mit $r_{\mathrm{D,px}}$ zum Skalierungsfaktor $S = \frac{200\,\mathrm{nm}}{27{,}6\,\mathrm{px}} \approx 7{,}25\,\frac{\mathrm{nm}}{\mathrm{px}}$ verknüpft werden.

Da die meisten Aufnahmen mit der gleichen Zoomstufe aufgenommen wurden, kann der Vorgang des Template-Matchings ausgelassen und die Zoomstufe direkt beim Funktionsaufruf der Oberflächenberechnungsfunktion einbezogen werden. In einer Lookup-Tabelle können Zoomstufe und der Skalierungsstufe festgehalten und bei Bedarf herangezogen werden. Eine Lookup-Tabelle, wie sie im Rahmen dieser Arbeit verwendet wurde, kann Tabelle 2.1 entnommen werden.

**Tabelle 2.1.:** Lookup Table für unterschiedliche Skalierungsfaktoren

| Zoomstufe | $S\left(\frac{\mu\mathrm{m}}{\mathrm{px}}\right)$ |
|---|---|
| 8000× | 0.0219 |
| 15000× | 0.0120 |
| 20000× | 0.0088 |
| 25000× | 0.0071 |
| 65000× | 0.0027 |
| $z$× | $\frac{175.2}{z}$ |

# 3 Kanten- und Kreisdetektion: Informationsextraktion

## 3.1 Hough-Transformation für Kreise

In diesem Kapitel wird die Hough-Transformation (HT) vorgestellt, welche zur genauen Bestimmung der Lochpositionen in der Templatfolie verwendet wird. Die HT gestattet die Lokalisation von parametrisierbaren Formen in Punktverteilungen [3].

Betrachtet wird der Einsatz der HT unter der Verwendung eines binären Kantenbildes eines Kreises mit bekanntem Radius $r_0$ (siehe Abb. 3.1, linker Graph). Ein solcher Kreis in 2D kann bekanntlich über zwei reellwertige Parameter $(a, b)$ beschrieben werden, beispielsweise in der Form

$$(x_i - a)^2 + (y_i - b)^2 = r_0^2. \tag{3.1}$$

Das Parameter-Paar $(a, b)$ entspricht dabei dem Mittelpunkt des betrachteten Kreises im sogenannten *Bildraum*, welcher von den Koordinaten $x$ und $y$ aufgespannt wird.
Ein Kreis, der durch drei gegebene Punkte $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ und $p_3 = (x_3, y_3)$ verläuft, muss daher folgende Gleichungen erfüllen:

$$\begin{aligned} (x_1 - a)^2 + (y_1 - b)^2 &= r_0^2 \\ (x_2 - a)^2 + (y_2 - b)^2 &= r_0^2 \\ (x_3 - a)^2 + (y_3 - b)^2 &= r_0^2 \end{aligned} \tag{3.2}$$

Ziel ist es nun, denjenigen Mittelpunkt $(a_0, b_0)$ zu finden, auf dessen zugehörigen Kreis möglichst viele Kantenpunkte $(x_i, y_i)$ zum Liegen kommen.
Im Wesentlichen arbeitet die HT mit einem Akkumulator-Array, dessen Einträge gezielt durch die „Stimmabgabe“ einzelner Kantenpunkte des Bildraumes erhöht werden. Jeder Punkt $(x, y)$ im Bildraum stimmt damit für eine Menge an möglichen Kreismittelpunkten ab. Diese Menge entspricht gerade einem Kreis der Parametrisierung

$$(a_i - x)^2 + (b_i - y)^2 = r_0^2 \tag{3.3}$$

im sogenannten *Parameterraum* (siehe Abb. 3.1, rechter Graph), welcher von den Koordinaten $a$ und $b$ aufgespannt wird. Der Akkumulator-Array rastert den Parameterraum zu einem Gitter mit klar definierten Zellen. Jede Zelle fungiert als Zähler der ihr zugeteilten Stimme. Zur Veranschaulichung wurden im Parameterraum diejenigen Zellen grau eingefärbt, die zwei oder mehr Stimmen der im Bildraum befindlichen Kantenpixel erhalten haben. Je dunkler die Zelle, desto mehr Stimmen hat diese erhalten. Jene Zellen mit den lokal höchsten Zählern werden als Positionen der gefundenen Kreise im Bildraum angenommen.

Für die eindeutige Lokalisation eines Kreises werden stets mindestens drei auf ihm liegende Punkte benötigt, so dass aus diesem Grund alle Zellen des Akkumulators mit weniger als drei

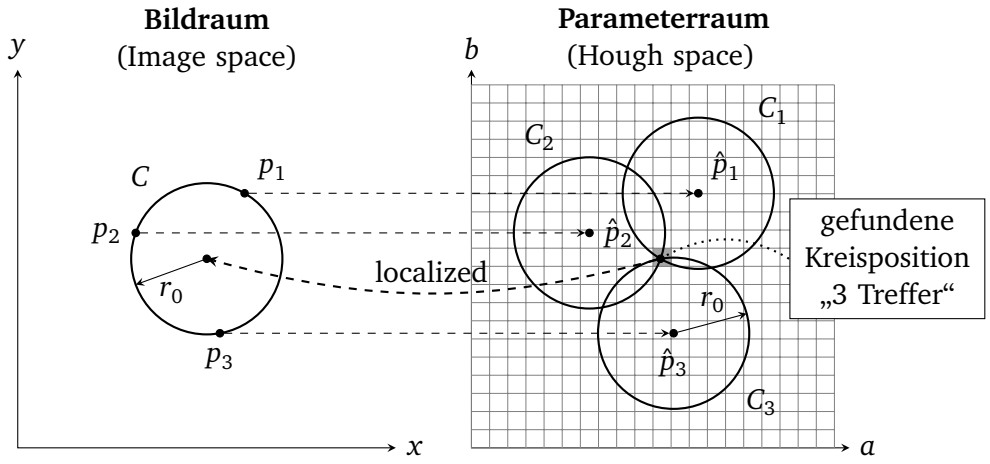



**Abbildung 3.1.:** Die Abbildung zeigt den Bild- und Parameterraum für einen Kreis $C$ und drei auf ihm liegende Punkte $p_1$, $p_2$ und $p_3$. Die Mittelpunkte aller Kreise, die durch einen gegebenen Bildpunkt $p_1 = (x_1, y_1)$ laufen, liegen selbst wieder auf einem Kreis $C_1$ um den Mittelpunkt $\hat{p}_1$. Analog gilt dieser Sachverhalt auch für die Kreise durch Bildpunkte $p_2$ und $p_3$. Letztendlich haben die Kreise einen gemeinsamen Schnittpunkt im echten Mittelpunkt des Kreises $C$ (ähnlich zu [3, S.168]). Das Akkumulator-Array ist Form eines Gitter über dem Parameterraum veranschaulicht worden. Jedes Kantenpixel kann dabei in einzelne Zellen „voten“, deren Zähler mit jeder Stimme inkrementiert wird. Anschließend muss das Gitter auf Maxima untersucht werden, um die Position der echten Kreismittelpunkte des Bildbereichs bestimmen zu können.

Stimmen ignoriert werden können. Letztendlich bleibt in diesem Fall lediglich Zelle $(a_0, b_0)$ mit drei Stimmen übrigen und kann damit als Position des Kreises $C$ angenommen werden.

Ein großer Vorteil des Akkumulator-Arrays ist die Möglichkeit, gezielt nach Kreisen hoher Kantenstärke suchen zu können, welche über eine auf den Akkumulator-Array angewandte Schwellenwertoperation realisiert werden kann. Dies ist mitunter ein Grund für die Robustheit der HT gegenüber verrauschten Bildern.

### Anwendung der Hough-Transformation in Software

Praktischerweise existieren in vielen Software-Umgebungen fertige Implementierungen der Hough-Transformation. Beispielsweise stellt MATLAB die Funktionen `hough`, `houghlines` und `houghpeaks` zur Verfügung, welche von `imfindcircles` zur Kreissuche eingesetzt werden. Aufgrund ihrer universellen Einsatzweise und Benutzerfreundlichkeit bildet `imfindcircles` für diese Arbeit eine zentrale Verwendung. Die genutzte Methodensignatur ist in Quelltext 3.1 abgedruckt.

**Quelltext 3.1:** Berechnung des Startwertes mithilfe binärer Suche

```
1 [centers,radii] = imfindcircles(A,radiusRange,objPolarity,method,sensitivity);
```

Eine Auflistung der nützlichen Parametern und deren Erläuterung folgen in Tabelle 3.1.

**Tabelle 3.1.:** Auflistung aller genutzten Parameter-Value-Paare der Funktion `imfindcircles`

| Parameter | Mögliche Werte | Erläuterung |
|---|---|---|
| A | Matrizen vom Datenformat single, double, int8, int16, logical, etc. | Das Bild, in welchem die Kreise detektiert werden sollen. Muss vorher über die Funktion `imread` eingelesen werden. |
| radiusRange | [min max] Ganzzahlige, numerische Zahlenwerte im Bereich von min bis max | Bezeichnet den Radius-Intervall in dem nach Kreisen gesucht werden soll. |
| 'ObjectPolarity' | 'bright' (default)<br>'dark' | Legt fest, ob die gesuchten Kreise dunkel oder hell im Bezug auf den Hintergrund sind. |
| 'Method' | 'PhaseCode' (default)<br>'TwoStage' | 'PhaseCode' [4] ist als Standardmethode eingestellt. |
| 'Sensitivity' | 0–1 (default: 0.85) | Der Sensitivitätswert beeinflusst den Akkumulator-Array und dessen Sensitivität gegenüber der Wahl der benötigten Stimmabgaben für eine erfolgreiche Kreisdetektion. Mit einem höheren Wert werden tendentiell mehr Kreise gefunden, allerdings steigt auch die Gefahr von falsch positiven Detektionen. |

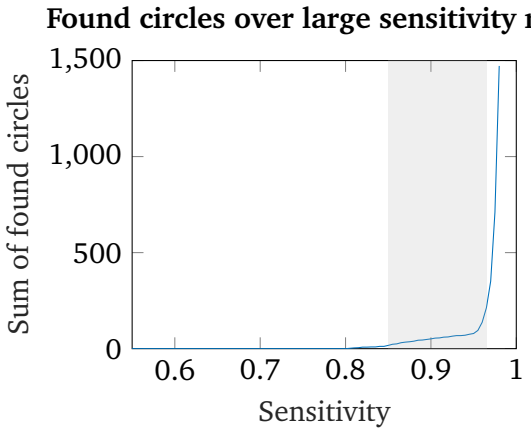
## 3.2 Anwendung von imfindcircles

### 3.2.1 Berechnung der Sensitivität

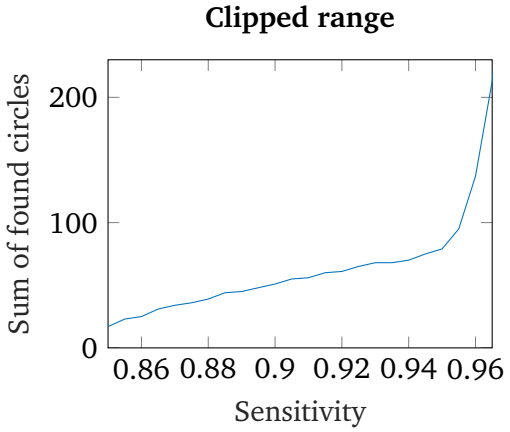
Der wichtigste Parameter zur Steuerung der Funktion `imfindcircle` ist `Sensitivity`. Bereits leichte Änderungen dieses Parameters haben weitreichende Folgen für das Ergebnis. Aus diesem Grund konzentriert sich diese Arbeit zu einem wesentlichen Teil auf die generische Bestimmung eines möglichst idealen Sensitivitätswerts mithilfe der ab Abschnitt 3.2.1 vorgestellten Suchmethode.

Die Sensitivität der Funktion `imfindcircle` kann über einen Wert zwischen 0 und 1 eingestellt werden. Allerdings werden Kreise in der Praxis erst ab einer Sensitivität von 0.5 und höher erkannt, so dass sich die Untersuchung eines geeigneten Werts unterhalb von 0.5 gespart werden kann. Aufgrund des hohen zeitlichen Aufwands, den `imfindcircles` für die Iteration über den gesamten Sensitivitätsbereich benötigt, ist das Auslassen von nicht relevanten Sensitivitätswerten von sehr hoher Relevanz. Im nächsten Abschnitt wird deshalb eine Methode erarbeitet, mit deren Hilfe ein sinnvoller Wertebereich (im Folgenden als *Arbeitsbereich* bezeichnet) ermittelt werden kann. Implementiert wird diese Methode in der Funktion `calculateSensitivityRange(Image, Radius)`, welche in vollständiger Form in Anhang A.2 nachgeschlagen werden kann.

Zu Beginn soll untersucht werden, wie sich die Anzahl gefundener Kreise mit Erhöhung der Sensitivität entwickelt. Dazu wird in Abb. 3.2a über einen großen Sensitivitätsbereich iteriert und die dabei gefundenene Kreise aufgetragen. Aus dem Plot wird ersichtlich, dass ab einer Sensitivität von etwa 0.96 die Anzahl gefundener Kreise exponentiell in die Höhe schießt. Auf der anderen Seite werden unter einem Sensitivitätswert von 0.8 keinerlei Kreise detektiert. Aus diesem Grund kann ein großer Teil des Wertebereichs ausgeklammert werden. Ein sinnvollerer Bereich für die praktische Untersuchung ist in Abb. 3.2b abgebildet.




**(a)** Iteration über einen sehr großen Sensitivitätsbereich. Der graue Bereich dient als Vorschlag für einen sinnvollen Arbeitsbereich, da sich in diesem Bereich in nachvollziehbarer Geschwindigkeit die Anzahl gefundener Kreise erhöht.




**(b)** Die Abbildung zeigt den Ausschnitt des Definitonsbereichs, welcher genau dem grauen Bereich der nebenstehenden Abbildung entspricht. Filigranere Verläufe und der Punkt des sprunghaften Anstiegs sind deutlich sichtbar.

**Abbildung 3.2.:** Es wird veranschaulicht, dass durch einen gezielten Bereichsausschnitt die Untersuchung auf Verlaufsmuster deutlich erleichtert werden kann.

Ein wesentlicher Anspruch an die Implementierung der Kreisdetektion ist die generische Bestimmung aller benötigten Parameter. Aus diesem Grund wird im nächsten Schritt die Möglichkeiten der automatisierten Bestimmung des Arbeitsbereiches untersucht.

### Bestimmung des Startwerts für den Arbeitsbereich

Mit einer binären Suche und einer vorgegebenen, minimalen Schrittweite $\epsilon$ wird derjenige Sensitivität als Startwert festgelegt, bei dem der erste Kreis detektiert wird. Je weniger Löcher in der Aufnahme zu sehen sind, desto größer kann $\epsilon$ gewählt werden. Die Implementierung der binären Suche und Ausgabe des Startwertes ist in Quelltext 3.2 abgedruckt.

**Quelltext 3.2:** Berechnung des Startwertes mithilfe binärer Suche

```
1 startPunkt = 0.5;
2 schrittWeite = 0.25;
3
4 epsilon = 0.01
```

```
5
6  while schrittWeite > epsilon
7         foundCircles = imfindcircles(image,[radius-floor(radius*0.19) ...
8                radius+floor(radius*0.19)],'ObjectPolarity','dark', ...
9                'Method','twostage','Sensitivity',startPunkt);
10        if ~isempty(foundCircles)
11               startPunkt = startPunkt - schrittWeite;
12        else
13               startPunkt = startPunkt + schrittWeite;
14        end
15
16        schrittWeite = schrittWeite/2;
17 end
18
19 startPunkt = round(startPunkt,2) + 0.01;
```

### Endwert des Arbeitsbereichs

Die Suche nach einem sinnvollen Endpunkt des Arbeitsbereich gestaltet sich wesentlich aufwändiger. Zum einen steigt der Rechen- und damit auch der Zeitaufwand enorm an, je mehr sich der Sensitivitätswert dem Maximalwert von 1 annähert. Dies ist im exponentiellen Anstieg der in Frage kommenden Kantenpixel begründet. Deshalb ist es von großem Interesse Berechnungen mit solchen Werten nach Möglichkeit zu meiden. Dies wird durch einen vorzeitigen Abbruch der Iterationen erreicht, wenn eine vorher festgelegte Maximalanzahl an Kreisen gefunden wurde (s. 3.3, Zeile 9).
Als Anhaltspunkt für den Endwert wird willkührlich der fünffache Wert des Medians[1] der Ableitungswerte der Einzelsummen gefundener Kreise als Schwellwert gewählt. Mit diesem Vorgehen wird der Arbeitsbereich rechtzeitig mit Beginn der explodierenden Werten abgegrenzt und gleichzeitig sichergestellt, dass der kritische Übergangspunkt zwischen dem Bereich leichten Anstiegs und demjenigen des explosiven Anstiegs noch mit in den Arbeitsbereich aufgenommen wird. Die Implementierung zur Endwert-Suche kann Quelltext 3.3 entnommen werden.

**Quelltext 3.3:** Berechnung des Endwertes

```
1  foundCircles = zeros(1,1000);
2  arrayCounter = 0;
3
4  for i = startPunkt: 0.01: 0.99
5         arrayCounter = arrayCounter + 1;
6         [centers,~] = imfindcircles(image,[radius-floor(radius*0.19) ...
7                radius+floor(radius*0.19)],'ObjectPolarity','dark', ...
8                'Method','twostage','Sensitivity',i);
9         if length(centers) < 500 % sinnvolle Maximalanzahl in einem Bild
10               foundCircles(arrayCounter) = length(centers);
11        else
12               break;
13        end
14 end
15
16 % Leite diskrete Funktion ab
```

[1] Im Gegensatz zum Durchschnittswert kommt der Median hervorragend mit starken Ausreißern zurecht, welche in diesem Fall gegen Ende des Arbeitsbereiches auftraten.

```
foundCirclesDiff = diff(foundCircles);

% Bestimme Median der Menge
medianValue = median(foundCirclesDiff);

% Bestimme die Stelle, an der der Anstieg der Anzahl der gefundenen
indexOfSensitivity = find(foundCirclesDiff>medianValue*5, 1, 'first');
endPunkt = startPunkt + 0.01* indexOfSensitivity;
```

Nach vielen Sensitivitätsiterationen mehrerer REM-Aufnahmen und deren Untersuchung auf die Entwicklung der Anzahl gefundener Kreise, wird folgende Hypothese aufgestellt: Genau bis zu jenem Punkt, an dem die Anzahl gefundener Kreise beginnt in die Höhe zu schießen, können noch echte Kreise gefunden werden. Gefundene Kreise über diesen Punkt hinaus, sind tendentiell nur Falschdetektion aufgrund des zu hoch eingestellten Sensitivitätswerts. Um diese Hypothese zu testen, werden in den folgenden Abschnitten mithilfe einer einfachen Kurvendiskussion eben jener kritischer Sensitivitätswert (im Folgenden als *Arbeitspunkt* $G_A$ bezeichnet) generisch bestimmt und die Ergebnisse nach dessen Einsatz in der Funktion `imfindcircles` bewertet.

### Suche des Arbeitspunktes $G_A$

Das bereits berechnete Array `foundCirclesDiff` liefert prägnante Punkte in Form von lokalen Minima, welche als Arbeitspunkte in Betracht gezogen werden können. Besonders interessant dabei ist das letzte lokale Minima, welches der letzte abgrenzbare Messpunkt im nicht-exponentiellen Bereich darstellt, bevor ein massiver Anstieg der Anzahl gefundener Kreise stattfindet (siehe Abb. 3.3).

Im nächsten Schritt werden die lokalen Minima des Arrays `foundCirclesDiff` berechnet. Diese treten auf, wenn die letzten Kreise einer bestimmten Kantenstärke gefunden werden. Besonders interessant ist das letzte auftretende Minima, da nach diesem die die Anzahl der gefundenen Kreise explodiert. Grund dafür ist die Berücksichtigung von kantenschwachen Strukturen sowie auftretendes Rauschen in der Berechnung.

Quelltext 3.4 dient als beispielhafte Implementierung der Arbeitspunktsuche.

**Quelltext 3.4:** Arbeitspunktsuche

```
foundCircles = zeros(1, 1000);
arrayCounter = 1;

for i=startPunkt:0.005:endPunkt
       [centers,~] = imfindcircles(image,[radius-floor(radius*0.19) ...
              radius+floor(radius*0.19)],'ObjectPolarity','dark', ...
              'Method','twostage','Sensitivity',i);
       foundCircles(arrayCounter) = length(centers);
       arrayCounter = arrayCounter + 1;
end

x = startPunkt+0.005:0.005:endPunkt;
y = foundCircles(1:arrayCounter-1);
differential = diff(y);

```

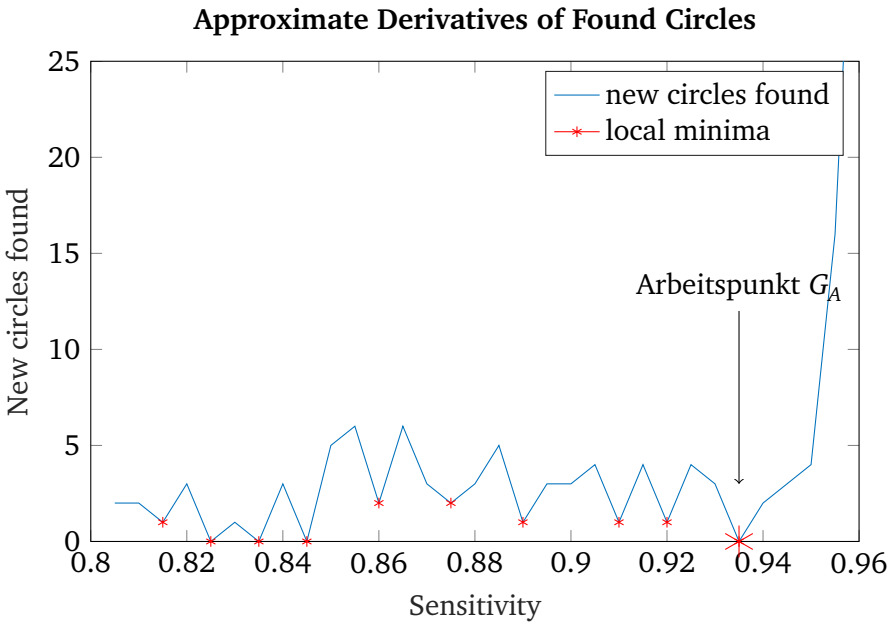



**Abbildung 3.3.:** Zu sehen ist das Array `foundCirclesDiff` als aufgetragener Funktionsgraph, welcher durch eine diskrete Ableitung des Graphen aus Abb. 3.2b produziert wurde. Zusätzlich sind lokale Minima als abgrenzbare, rote Punkte aufgetragen worden. Besonderes Augenmerk liegt dabei auf dem letzten lokalen Minima an der Stelle 0.935, welches als Übergangspunkt zwischen dem Bereich sinnvoller, langsam steigender Kreisdetektionen und jenem explodierender, falsch positiver Detektionen dienen soll.

```
16 indicesOfLocalMinima = islocalmin(differential);
17 localMinima = x(indicesOfLocalMinima);
18
19 sensitivityPoint = localMinima(end);
```

Als Demonstration des durch den in Abb. 3.3 gefundenen Arbeitspunkt, soll eine Kreisdetektion mit dem ermittelten Sensitivitätswert von 0.935 duchgeführt werden.

## 3.3 Kantendetektion

Die stärksten Kanten in den von uns betrachteten REM-Bildern sind diejenigen, welche die Einschusslöcher der Templatfolie umreißen. Somit liegt es nahe, dass man mit wenig Rechenaufwand eine gezielte Kantedetektion durchführen kann, um jene Kanten zu extrahieren und anhand dieser die aufgespannten Mantelflächen berechnet. Probleme können falsch positiv detektierte Kanten verursachen, wenn nicht orthogonal geschossene Löcher untersucht werden. Hier kann unter Umständen der Fall eintreten, dass unvollständige Kreise oder Verschmierungen (siehe Abb. 1.1b) zutage treten, welche eine präzise Umrissabschätzung erschweren.

### Grundlegender Algorithmus zur Detektion von Kanten

1. Weichzeichnung des Bildes zur Herausfilterung von Rauschen

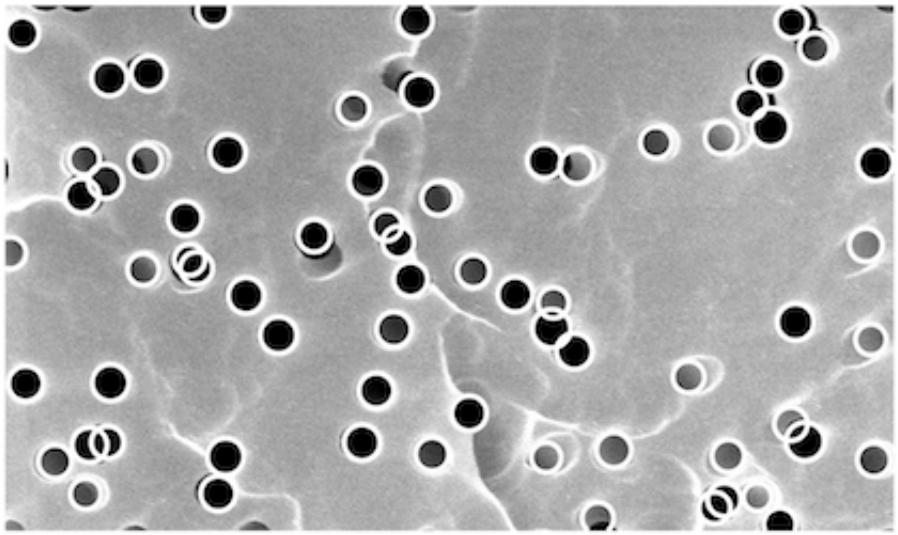


**Abbildung 3.4.:** Resultat einer Kreisdetektion. Gefundene Kreise sind mit einer dicken, weißen Umfangslinie an die entsprechenden Stellen gerendert worden. Mit zwei Ausnahmen wurde jedes einzelne Loch aufgespührt. Falschdetektionen (false positives) traten nicht auf.

2. Berechnung des Gradientenbildes
3. Anwendung einer Schwellenwertoperation, um signifikante Gradienten zu finden
4. „Ausdünnung“ von Kanten, um Konturen mit einer Dicke von einem Pixel zu erhalten und damit eine genaue Lokalisation zu ermöglichen
5. Optional: Diese Kantenpixel miteinander verbinden, um eine durchgehende Kontur zu erhalten

### 3.3.1 Canny Algorithmus

Wie bereits im Einführungskapitel beschrieben, wird aufgrund seiner hervorragenden Performance der Canny-Filter für diese Aufgabe auserwählt. Er wird im Folgenden nur grob beschrieben, da auch hier eine fertige Implementierung seitens MATLAB verwendet wird.
Der Canny-Filter arbeitet mit der zweiten Ableitung und nimmt zusätzlich die Positionen der Nulldurchgänge der zweiten Ableitung als Kantenposition hinzu. Um die Anfälligkeit gegenüber Rauschen zu mindern, wird ein geeigneter Glättungsfilter, wie beispielsweise der Laplacian-of-Gaussian-Operator [10].

#### Anschaulicher Algorithmus

1. Bild mit Gauss-Filter filtern:
2. Finde Betrag und Orientierung der Kanten
3. Non-Maximum suppression: Ausdünnung von Multi-Pixel-Kanten zu 1-Pixel-Kanten (entspricht einer Breite von einem Pixel)

4. Verbindung und Schwellenwertbildung von Kantenpixeln: a) Bestimmung zweier Schwellwerte: niedrig und hoch b) Nutzung des hohen Schwellwertes, um Konturen hoher Kantenstärke zu bilden und c) die Fortführung dieser Konturen mithilfe des niedrigen Schwellwertes (Details zu diesem Vorgehen finden sich im nächsten Paragraphen zur Schwellenwert Hysterese)

#### Canny Schwellenwert Hysterese

Im Wesentlichen werden zur Kantenbildung vom Canny-Algorithmus folgende Arbeitsschritte durchgeführt:

1. Wende Kantenfilter mit hohem Schwellenwert an, um besonders ausgeprägte Kanten zu finden
2. Verbinde die gefundenen, starken Kantenpixel miteinander, um starke Konturen zu bilden
3. Wende nun Kantenfilter mit einem niedrigen Schwellenwert an, um schwache, aber plausible Kanten zu finden
4. Erweitere nun die starken Kanten in Richtung der schwachen Kantenpixel

Ob das Ergebnis zufriedenstellend ist, lässt sich pauschal nicht vorhersagen und bedarf in der Regel einer experimentellen Versuchsreihe.
Da die reine Kantendetektion zur Oberflächenbestimmung in dieser Arbeit nur eine mögliche Alternative darstellt, findet eine weitergehende Vertiefung der Kantendetektion im Rahmen dieser Arbeit nicht statt. Die Kanten für die in Kapitel 4.2.1 vorgestellte pixelbasierte Oberflächenberechnung liefert die in MATLAB vorimplementierte Version des Canny-Algorithmus in Quelltext 3.5.

**Quelltext 3.5:** Methodensignatur der Funktion `edge` unter Auswahl des Canny Filters

```
1 BW = edge(Image,'canny',THRESH,SIGMA);
```

Beispielhafte Kantendetektionen mithilfe vom Canny-Algorithmus können Abbildungen 1.1a und 1.2a entnommen werden.

# 4 Umfangs- und Oberflächenberechnung

In diesem Kapitel werden zwei Möglichkeiten der Oberflächenberechnung vorgestellt. Hierbei werden zwei wichtige Vereinfachungen für die Beschaffenheit der Nanoröhrchen getroffen:

1. Das Volumen eines einzelnen Röhrchens entspricht einem geometrischen Zylinder
2. Jedes Röhrchen steht orthogonal zur Grundfläche und stellt damit einen in $z$-Richtung invarianten Körper dar

Auf diesen beiden Annahmen beruhend, kann die Mantelfläche eines einzelnen Nanoröhrchens direkt mit $2\pi rh$ angegeben werden. Dabei entspricht $r$ dem Radius der Zylindergrundfläche und $h$ der Zylinderhöhe. In der Praxis kann es jedoch passieren, dass sich in einem Drahtarray mehrere Röhrchen an ihrer Wurzel überschneiden und damit eine einfache Berechnung der Oberfläche mit obiger Formel nicht möglich ist. Im folgenden Abschnitt 4.1 wird auf die Berechnung einer durch die Überlappung mehrerer Röhren entstandenen Umfangslinie näher eingegangen.

## 4.1 Geometrische Umfangsberechnung

Dieser Abschnitt befasst sich mit dem mit dem mathematischen Hintergrund des geometrischen Problems der Berechnung des Umfangs von beliebig vielen, überlappenden Kreisen und dessen Lösung mithilfe eines Algorithmus am Ende des Kapitels.

### 4.1.1 Überlappung zweier Kreise

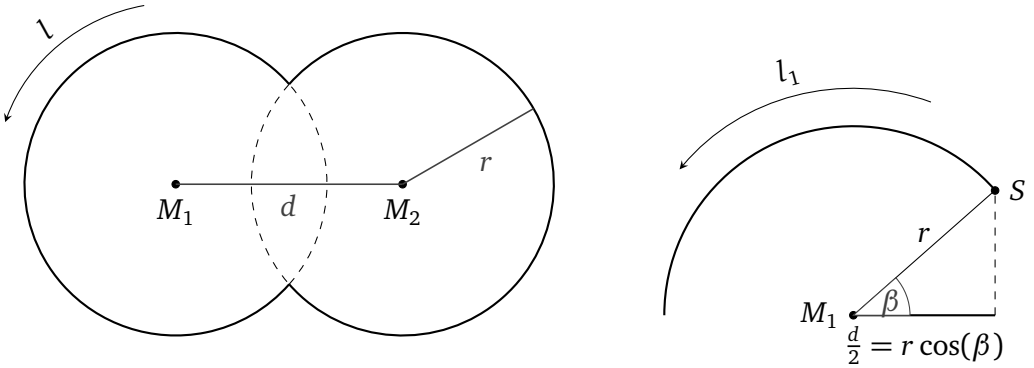
Zu Beginn soll sich die Untersuchung lediglich auf den Umfang zweier sich schneidender Kreise beziehen und erst in einem weiteren Schritt auf die Lösung des Problems von beliebig vielen Überschneidungen an beliebigen Positionen in Abschnitt 4.1.2 ausgeweitet werden.

#### Problemstellung

Gegeben seien zwei Kreise mit dem Radius $r$, deren Mittelpunkte $M_1$ und $M_2$ sich im Abstand $d$ zueinander befinden. Dabei sollen diese auf ihren gemeinsamen Umfang $l$ untersucht werden. Wird $d < 2r$ gewählt, so findet eine Überschneidung statt. Dadurch reduziert sich der Gesamtumfang im Vergleich zu zwei Einzelkreisen (gestrichelte Linie in Abb. 4.1a).

#### Mathematische Herleitung

Für die Berechnung des zum Umfang beitragenden Bogenstücks wird der Winkel $\beta$, definiert, dessen Schenkel die Strecken $\overline{M_1M_2}$ und $\overline{M_1S}$ bilden. $S$ ist hierbei einer der Schnittpunkte der überlappenden Kreise. Dabei gilt gemäß Abbildung 4.1b folgender Zusammenhang:




**(a)** Zwei überschneidende Kreis und deren resultierende Umfangslinie

**(b)** Zusammenhang zwischen Winkel $\beta$ und dem Abstand $d$ beider Kreismittelpunkte

**Abbildung 4.1.:** Weitere Größen und deren Orientierung im Raum

$$\beta = \cos^{-1}\left(\frac{d}{2r}\right) \tag{4.1}$$

Mit dem berechneten Winkel $\beta$ kann nun der Teil des Kreisbogens bestimmt werden, der nach der Überschneidung übrig bleibt. Dazu wird die Bogenlänge des verbleibenden Kreissegments $l_1$ berechnet, wie es folgende Gleichung zeigt:

$$\begin{aligned} l_1 &= 2\pi r\left(1-\frac{\beta}{\pi}\right) \\ &= 2\pi r\left(1-\frac{\cos^{-1}\left(\frac{d}{2r}\right)}{\pi}\right) \end{aligned} \tag{4.2}$$

Weiterhin gilt für den Gesamtumfang die resultierende Umfangslinie $l$:

$$l = 2l_1 = 4\pi r\left(1-\frac{\cos^{-1}\left(\frac{d}{2r}\right)}{\pi}\right). \tag{4.3}$$

Aufgrund der Spiegelsymmetrie gelten die gleichen Zusammenhänge auch für den Kreis $C_2$ um den Mittelpunkt $M_2$ und münden in Gleichung 4.3

### 4.1.2 Überlappung beliebiger Anzahl Kreise

Gegeben sei dabei eine beliebige Anordung von $n$ Kreisen $C_1, C_2, ..., C_n$ mit den Mittelpunkten $M_1, M_2, ..., M_n$. Eine beispielhafte Anordnung solcher Kreise kann Abbildung 4.2 entnommen werden.

Zu beachten ist hier der resultierende Umfang der Anordnung (durchgezogene, dicke Linie). Diesen gilt es im Folgenden schrittweise zu berechnen.

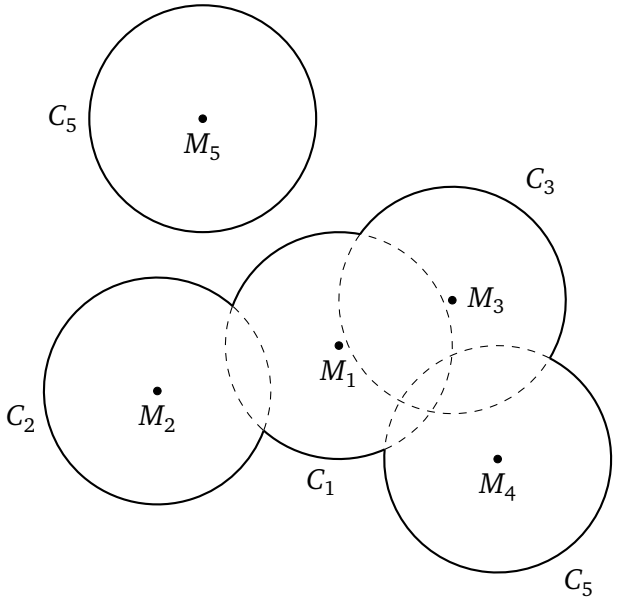



**Abbildung 4.2.:** Beispielhafte Anordnung mehrerer Kreise

## Definitionen und mathematische Zusammenhänge

Der am Ende dieses Abschnitts vorgestellte Algorithmus bestimmt in Form von Adjazenzmatrizen die paarweisen Abstände, Abstandsvektoren und Schnittpunkte aller Kreise zueinander. Anhand der Orientierung und Lage dieser Größen kann über Vergleichsoperationen der nicht überdeckte Teil des Bogens jedes einzelnen Kreises bestimmt werden. Dieser wird zur letztendlichen Berechnung der Oberfläche herangezogen. Veranschaulicht wird die Bestimmung der verbleibenden Kreisbogensegmente durch ein sogenanntes Spurendiagramm (Abb. 4.5a).

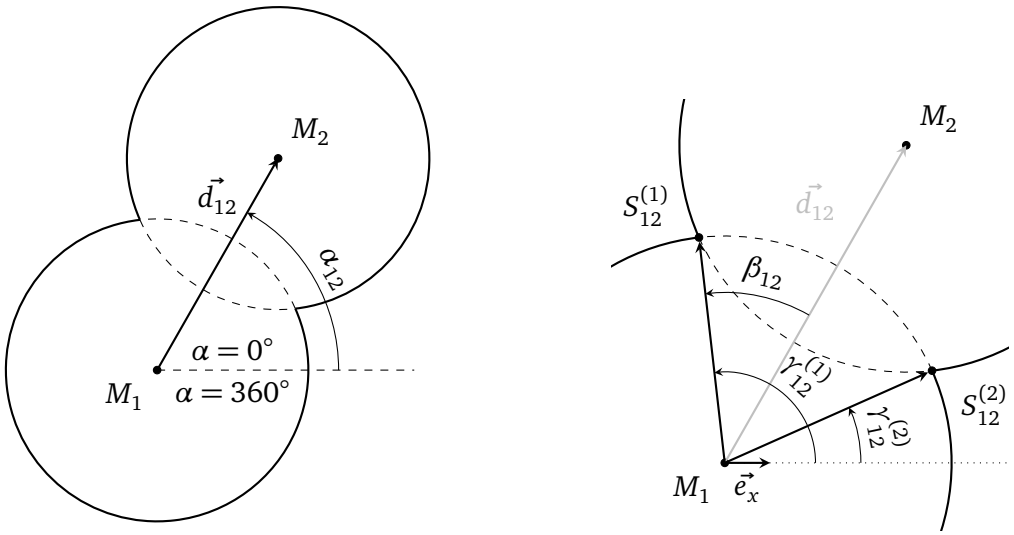
### Berechnung der Abstandsvektoren und deren Orientierung

Fasst man die Kreismittelpunkte als Knoten und die Abstandsvektoren als Gewichte der Kanten auf, kann zu jeder beliegen Kreiskonfiguration eine Adjazenzmatrix aufgestellt werden, welche alle benötigten Größen in kompakter Form speichern kann. In einem weiteren Schritt können aus dieser Adjazenzmatrix die exakten Schnittpunkte der Kreise ermittelt und zur Berechnung der nicht überdeckten Kreisbogen herangezogen werden.
Die Abstandsvektoren werden in einer Adjazenzmatrix $D = \left[\vec{d}_{ij}\right]$ durch ihre Einträge definiert als [Quellenverweis]

$$\vec{d}_{ij} = \begin{cases} \vec{m}_j - \vec{m}_i & \text{falls } i \neq j \\ 0 & \text{sonst.} \end{cases} \tag{4.4}$$

Hierbei ist $\vec{d}_{ij}$ der Abstandsvektor von Raumpunkt $M_i$ des Kreises $C_i$ zum Raumpunkt $M_j$ des Kreises $C_j$. Die Vektoren $\vec{m}_i$ und $\vec{m}_j$ stellen dabei die Ortsvektoren der Kreismittelpunkte dar.

Darauf aufbauend wird nun für jeden Kreis eine Orientierung der von ihm ausgehenden Abstandsvektoren definiert. Die Orientierung eines Abstandsvektors wird dabei wie in Abbildung 4.3a in einem in Gleichung 4.5 definierten Winkel $\alpha_{ij}$ gespeichert. Wir fassen diesen Winkel als denjenigen auf, der von $\vec{e_x}$ (welche der gestrichelten Horizontalen durch den Punkt $M_1$ entspricht) und dem Distanzvektor $\vec{d_{ij}}$ aufgespannt wird. Er verläuft im mathematisch positiven Sinn beginnend von der $x$-Achse eines kartesischen Koordinatensystems, dessen Ursprung genau im Mittelpunkt $M_i$ des betreffenden Kreises $C_i$ liegt, vom ersten bis zum vierten Quadranten.




**(a)** Orientierung eines Abstandsvektors **(b)** Kreis-Schnittpunkte und ihre Lage

**Abbildung 4.3.:** Weitere Größen und deren Orientierung im Raum

Um den definierten Winkel $\alpha_{ij}$ zu erhalten, wird eine Schnittwinkel-Berechnung [11] gemäß Gleichung 4.5 des jeweiligen Distanzvektors $d_{ij}$ und des $\vec{e_x}$-Einheitsvektors durchgeführt.

$$\alpha_{ij} := \angle(\vec{d_{ij}}, \vec{e_x}) = \cos^{-1} \frac{\left|\vec{d_{ij}} \cdot \vec{e_x}\right|}{\left|\vec{d_{ij}}\right| \cdot \left|\vec{e_x}\right|} \tag{4.5}$$

Dabei wird eine weitere, symmetrische Adjazenzmatrix $D_\alpha = \left[\alpha_{ij}\right]$ definiert, welche in analoger Indizierung die Orientierung der Abstandsvektoren speichert. Wünschenswert ist hierbei ein Wertebereich von $[0, 360]$ für den Winkel $\alpha_{ij}$. Allerdings stellt die Schnittwinkelformel 4.5 nur einen Bereich von $[0, 180]$ zur Verfügung, so dass Gleichung 4.5 zu folgender Form erweitert werden muss:

$$\alpha_{ij} = \begin{cases} \cos^{-1} \frac{\left|\vec{d_{ij}} \cdot \vec{e_x}\right|}{\left|\vec{d_{ij}}\right| \cdot \left|\vec{e_x}\right|} & \vec{d_{ij,y}} \geq 0 \\ 360° - \cos^{-1} \frac{\left|\vec{d_{ij}} \cdot \vec{e_x}\right|}{\left|\vec{d_{ij}}\right| \cdot \left|\vec{e_x}\right|} & \vec{d_{ij,y}} < 0 \end{cases} \tag{4.6}$$

Ist die $y$-Komponente des Distanzvektors $\vec{d}_{ij}$ kleiner Null, so liegt ein Vektor mit negativer Steigung vor und dem damit verbundenen überstumpfen Winkel $\alpha$. Da 4.5 in diesem Fall den spitzen Winkel beider Schnittgeraden berechnet, muss dieser von 360° subtrahiert werden, um den gewünschten Winkel $a_{ij}$ zu erhalten.

### Lage der Schnittpunkte

Mithilfe der Lage der Abstandsvektoren lassen sich nun ebenfalls auch die Orientierungen der Schnittpunkte $S_{ij}$ festhalten. Die genauen Positionen der Schnittpunkte des Kreises $C_i$ mit dem Kreis $C_j$ können dabei beispielhaft Abbildung 4.3b entnommen werden.
Mit Gleichung 4.1 kann der Winkel $\beta_{ij}$ mit

$$\beta_{ij} = \cos^{-1} \frac{\left|\vec{d}_{ij}\right|}{2r} \tag{4.7}$$

direkt angegeben werden.

Weiterhin ist zu beachten, dass sich schneidende Kreise stets zwei Schnittpunkte miteinander haben. Aus diesem Grund wird für jedes überlappende Kreispaar in Gleichung 4.8 ein Bogen $b_{ij}$ definiert, welcher den Anfangs- und Endwert des nicht verdeckten Kreisbogens als abgeschlossenes Intervall $[\gamma_{ij}^{(1)}, \gamma_{ij}^{(2)}]$ speichert. Die Grenzen dieser Menge entsprechen dabei gerade den Winkeln, an denen sich die Schnittpunkte schneidenden Kreise befinden. Über den Modulo-Operator wird sichergestellt, dass Unter- und Überschreitungen des gültigen Winkelbereichs $[0, 360]$ auf den korrekten Wert abgebildet werden.

$$b_{ij} := \left[\gamma_{ij}^{(1)}, \gamma_{ij}^{(2)}\right] = \left[(\alpha_{ij} + \beta_{ij}) \pmod{360}, (\alpha_{ij} - \beta_{ij}) \pmod{360}\right] \tag{4.8}$$

Alle Intervalle $b_{ij}$ werden wiederum in einer neuen Adjazenzmatrix $A = \left[b_{ij}\right]$ gespeichert, deren Spalte $i$ dabei als eine Menge $c_i$ von abgeschlossenenen Intervallen aller Bögen auf dem Kreis $C_i$ gilt. In Gleichung 4.9 wird der Begriff einer Intervallmenge näher definiert, um im kommenden Abschnitt eine mathematische Schnittmenge mehrerer Bögen bzw. Intervalle bilden zu können.

$$\begin{aligned} c_i &= \left\{b_1, b_2, ..., b_k\right\} \\ &= \left\{\left[\gamma_1^{(1)}, \gamma_1^{(2)}\right], \left[\gamma_2^{(1)}, \gamma_2^{(2)}\right], ..., \left[\gamma_k^{(1)}, \gamma_k^{(2)}\right]\right\}, \quad \gamma \in [0, 360) \end{aligned} \tag{4.9}$$

In Abbildung 4.4 ist zu sehen, dass der Schnittpunkt $S_{ij}^{(1)}$, dessen Position über den Winkel $\gamma_{12}^{(1)}$ angegeben ist, der ersten Begrenzung des Intervalls $a_{12}$ entspricht. Dieser Logik folgend wird dieses Intervall über den zweiten Schnittpunkt $S_{12}^{(2)}$ und den Winkel $\gamma_{12}^{(2)}$ geschlossen. Auf diese Weise können wir aus jedem Distanzvektor ein Schnittpunkt-Paar ermitteln, deren zugehörigen Winkel die Begrenzungen eines Intervalls bilden.

*Anmerkung:* Zu beachten ist die Möglichkeit, dass die Intervallgrenzen des Bogens $a_{ij}$ auch den Übergang des Winkels von 0° zu 360° beinhalten und damit der Endpunkt $\gamma_{ij}^{(2)}$ nominell

kleiner sein kann als der Startpunkt $\gamma_{ij}^{(1)}$. Beispielsweise ist $b_{12} = [270, 30]$ ein gültiger Bogenintervall.
In der Implementierung muss dieser Umstand in einem gesonderten Fall berücksichtigt werden.

#### Zusammenlegung von Bögen

Sobald alle paarweisen überlappten und nicht überlappten Bogensegmente in Form von Intervallen auf den Kreisen bestimmt und in der Matrix $A$ gespeichert sind, kann die Zusammenführung dieser spaltenweise im Sinne des dazugehörigen Kreises durchgeführt werden.

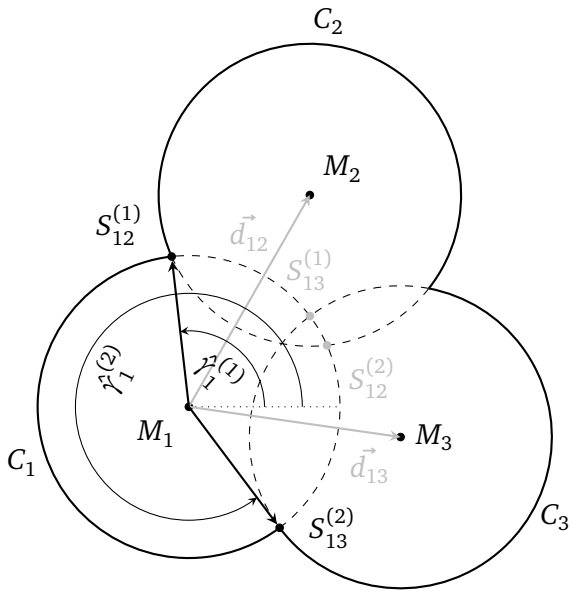



**Abbildung 4.4.:** Zusammenführung der Bögen $\overline{S_{12}^{(1)}S_{12}^{(2)}}$ und $\overline{S_{13}^{(1)}S_{13}^{(2)}}$ auf Kreis $C_1$ (durchgezogene als auch gestrichelte Linien bilden gemeinsam einen Bogen)

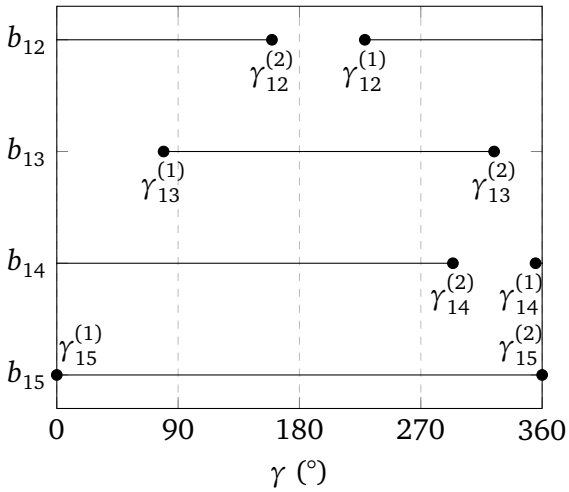
Für jeden Kreis liegen nun in der Matrix $A$ diejenigen Bögen vor, die nach der Überschneidung mit jedem Nachbarkreis übrig bleiben. Der auf $C_1$ verlaufende Kreisbogen des Kreispaares $(C_1, C_2)$, welcher von $S_{12}^{(1)}$ nach $S_{12}^{(2)}$ verläuft, bildet mit dem Kreisbogen des Kreispaares $(C_1, C_3)$ eine Schnittmenge und bildet einen neuen auf $C_1$ verlaufenden Kreisbogen, welcher von $S_{12}^{(1)}$ nach $S_{13}^{(2)}$ verläuft. Abbildung 4.4 verdeutlicht diesen Sachverhalt.
Da jeder Kreis nach dieser Prozedur eine unterschiedliche Anzahl an nicht überlappten Bogensegmenten besitzt, beinhaltet nach Definition 4.10 jede neue Intervallmenge $\hat{c}_i$ eine individuelle Anzahl $l$ an Intervallen.

$$\begin{aligned}\hat{c}_i &= b_1 \cap b_2 \cap \ldots \cap b_k, \quad b_i \in c_i \\ &= \left\{\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_l\right\} \\ &= \left\{\left[\hat{\gamma}_1^{(1)}, \hat{\gamma}_1^{(2)}\right], \left[\hat{\gamma}_2^{(1)}, \hat{\gamma}_2^{(2)}\right], \ldots, \left[\hat{\gamma}_l^{(1)}, \hat{\gamma}_l^{(2)}\right]\right\}, \quad \hat{\gamma} \in [0, 360)\end{aligned} \tag{4.10}$$

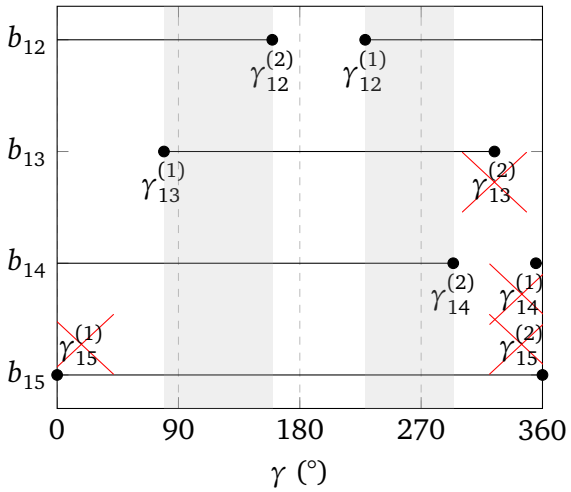
Die schrittweise Berechnung dieser Schnittmenge wird in Algorithmus 1 beschrieben. Nach dessen Anwendung auf die Bogenintervalle $b_i$ jeden Kreises, gibt dieser die neuen Bogenintervalle $\hat{b}_i$ innerhalb der Menge $\hat{c}_i$ als Prozedurausgabe *MergedArcsOnCirlce* aus und speichert

sie in einer neuen Liste zur weiteren Verarbeitung.

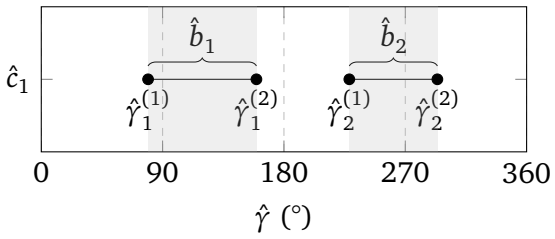
Abbildung 4.5 stellt den Zusammenhang zwischen den obigen mathematischen Definitionen und dem Algorithmus 1 her und zeigt die schrittweise Ausführung exemplarisch am Kreis $C_1$ aus Abbildung 4.2.



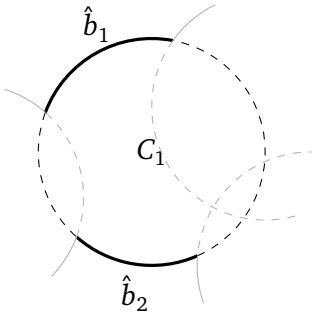

**(a)** Schritt 1: Start- und Endpunkte der Bögen $a_{ij}$ sammeln




**(b)** Schritt 2: Entferne Start- und Endpunkte aus überdeckten Bereichen




**(c)** Schritt 3: Setze übriggebliebene Punkte zu neuen Bögen $\hat{b}_1$ und $\hat{b}_2$ zusammen




**(d)** Resultierende Bögen $\hat{b}_1 = \left[\hat{\gamma}_1^{(1)}, \hat{\gamma}_1^{(2)}\right]$ und $\hat{b}_2 = \left[\hat{\gamma}_2^{(1)}, \hat{\gamma}_2^{(2)}\right]$ auf Kreis $C_1$

**Abbildung 4.5.:** Schrittweise Ausführung des Algorithmus 1 für den Kreis $C_1$ aus Abb. 4.2

Die Kurven, die sich in Abb. 4.5b direkt vor dem grau unterlegten Bereich befinden, bilden die nicht überdeckten Bogensegmente des Kreises $C_1$ ab. Dieser Bereich wird in *Step 2* in Alg. 1 berechnet, um anschließend außerhalb liegende Schnittpunkte zu eliminieren. Formal kann das graue Areal auch als die Schnittmenge der Definitionsbereiche aller Mengenfunktionen $a_{ij}$ gesehen werden. Bildlich entspricht das genau den Stellen, in denen alle „Bogenspuren" direkt übereinander liegen. Je größer dieser Bereich ist, desto weniger wird der betrachtete Kreis von anderen Kreisen überlappt. Umgekehrt gilt bei gänzlicher Abwesenheit dieser Bereiche eine komplette Überdeckung des Kreises, so dass dieser keinen Beitrag zum Umfang und damit der Oberfläche leistet.

**Algorithm 1** Schnittmengenbildung aller Bögen eines Kreises

```
1:  procedure MergeArcsForCircle(circle)
2:      Step 1 – collect start and end points of all arcs
3:      Create an empty list StartPointsOnCircle ← {}
4:      Create an empty list EndPointsOnCircle ← {}
5:      for all arcs (γ^(1), γ^(2)) in circle do
6:          Add γ^(1) to StartPointsOnCircle
7:          Add γ^(2) to EndPointsOnCircle
8:      end for
9:      Sort StartPointsOnCircle and EndPointsOnCircle by ascending order
10:     Remove duplicates in StartPointsOnCircle and EndPointsOnCircle
11:
12:     Step 2 – remove start and end points which are located in overlapped areas
13:     for all arcs (γ^(1), γ^(2)) in circle do
14:         if γ^(2) − γ^(1) > 0 then
15:             Delete all values s_i from StartPointsOnCircle if s_i < γ^(1) or s_i > γ^(2)
16:             Delete all values e_i from EndPointsOnCircle if e_i < γ^(1) or e_i > γ^(2)
17:         else
18:             Delete all values s_i from StartPointsOnCircle if s_i < γ^(1) and s_i > γ^(2)
19:             Delete all values e_i from EndPointsOnCircle if e_i < γ^(1) and e_i > γ^(2)
20:         end if
21:     end for
22:
23:     Step 3 – stitch start and end points in the right way together
24:     Create an empty list MergedArcsOnCircle ← {}
25:     Set up a counter k ← 1
26:     for all values γ^(1) in StartPointsOnCircle do
27:         Iterate over EndPointsOnCircle, find first value e which meets e > γ^(1)
28:         if e ≠ 0 then
29:             Create arc b̂_i = (γ̂_i^(1), γ̂_i^(2)) ← (γ^(1), e)
30:             MergedArcsOnCircle(k) ← b̂_i
31:             Increment k
32:         else
33:             Create arc b̂_i = (γ̂_i^(1), γ̂_i^(2)) ← (γ^(1), EndPointsOnCircle(1))
34:             MergedArcsOnCircle(k) ← b̂_i
35:         end if
36:     end for
37:     return MergedArcsOnCircle
38: end procedure
```

### Berechnung des Umfangs

Um aus den im vorherigen Kapitel berechneten Intervallmengen $\hat{c}_1, \hat{c}_2, ..., \hat{c}_n$ die Umfangslinie der Kreiskonfiguration $C_1, C_2, ..., C_n$ $l$ zur Bestimmung der Mantelfläche $F$ zu berechnen, werden im ersten Schritt die Längen der Intervalle $\hat{b_1}, \hat{b_2}, ..., \hat{b_l}$ über Gleichung 4.11 bestimmt.

$$l_{\hat{b}_j} := \begin{cases} \hat{\gamma}_j^{(2)} - \hat{\gamma}_j^{(1)} & \text{für } \hat{\gamma}_j^{(2)} \geq \hat{\gamma}_j^{(1)} \\ 360° - \left(\hat{\gamma}_j^{(1)} - \hat{\gamma}_j^{(2)}\right) & \text{für } \hat{\gamma}_j^{(2)} < \hat{\gamma}_j^{(1)} \end{cases} \tag{4.11}$$

Anschließend werden diese Intervalllängen zur Gesamtintervalllänge $l_{\hat{c}_i} = \sum_{j=1}^{l} l_{\hat{b}_j}$ aufsummiert.
Für die letztendliche Berechnung der nicht überdeckten Umfangslinie $l_i$ des Kreises $C_i$ wird diese Gesamtintervalllänge mit dem Umrechnungsfaktor $\frac{\pi}{180°}$ ins Bogenmaß überführt und mit dem Radius $r_i$ des dazugehörigen Kreises $C_i$ gewichtet.

$$l_i = l_{\hat{c}_i} \frac{\pi}{180°} r_i \tag{4.12}$$

Die Aufsummierung aller einzelnen Umfangslinien der Kreise $C_1, C_2, ..., C_n$ resultiert in der Gesamtumfangslinie $l = \sum_{i=1}^{n} l_i$. Schlussendlich kann in Gleichung 4.13 die Mantelfläche $F$ über die Drahthöhe $h$ bestimmt werden.

$$F = lh \tag{4.13}$$

Mithilfe eines Skalierungsfaktors, welcher die Länge eines Pixels in Mikrometern angibt, lässt sich letztendlich der reale Umfang bestimmen.

### 4.1.3 Implementierung

Die Implementierung gestaltet sich bei der Arbeit mit MATLAB besonders einfach. MATLAB stellt im Rahmen seiner *Image Processing Toolbox* eine umfangreiche Sammlung von Funktionen zur Bildverarbeitung zur Verfügung, welche in dieser Arbeit Verwendung finden.

#### Distanzvektoren und -winkel

**Aufstellen einer Adjazenzmatrix zur Speicherung von Distanzvektoren**

Viele der von uns in diesem Abschnitt benötigten Operationen können mit hauseigenen Funktionen der MATLAB-Umgebung gewonnen werden.

Die Dimensionen der Adjazenzmatrix $D$ (siehe Gleichung (4.14)) können mithilfe `pdist`-Funktion bestimmt werden:

**Quelltext 4.1:** Dimensionsbestimmung der Abstandsvektorenmatrix $D$

```
1 [D_rows, D_columns] = size(squareform(pdist(centers)));
```

Als Eingabeparameter für den Aufruf in Quelltext 4.1 dient `centers`, welche die Positionen der gefundenen Kreise angibt und von der vorangegangenen Funktion `imfindcircles` zurückgegeben wurde.

Die Bestimmung der Distanzvektoren $\vec{d}_{ij}$ erfolgt über folgende Zeilen:

**Quelltext 4.2:** Bestimmung der Distanzvektoren $\vec{d}_{ij}$

```
center1 = centers(i,:);
center2 = centers(j,:);
distanceVector = center2 - center1;
```

Durch Subtraktion der beiden Ortsvektoren $\vec{m}_i$ und $\vec{m}_j$ erhält man den Vektor $\vec{d}_{ij}$ zwischen den Kreismittelpunkten $M_i$ und $M_j$.

### Aufstellen einer Adjazenzmatrix zur Speicherung der Winkel $\alpha_{ij}$

Für die Bestimmung der Orientierung der Distanzvektoren (siehe Gleichung 4.5 über die Winkel $\alpha_{ij}$ wird der Einheitsvektor $\vec{e}_x$ benötigt (Quelltext 4.3, Zeile 1). Die eigentliche Schnittwinkelberechnung folgt in der nächsten Zeile desselben Quelltextes.

**Quelltext 4.3:** Bestimmung des Schnittwinkels $\alpha_{ij}$ zwischen $\vec{d}_{ij}$ und $\vec{e}_x$

```
x_e = [1 0];
angle = acosd(min(1,max(-1, x_e(:).' * distanceVector(:) ...
              / norm(x_e) / norm(distanceVector) )));
```

Um den Wertebereich von $[0, 180]$ auf $[0, 360]$ zu erweitern (siehe Gleichung 4.6), müssen die Winkel $\alpha_{ij}$ anhand der Steigung des Distanzvektors $\vec{d}_{ij}$ (betrachtet wird hier die $y$-Komponente) angepasst werden:

**Quelltext 4.4:** Erweiterung des Bildbereichs der Schnittwinkelformel 4.5

```
if distanceVector(2) > 0
       angle = 360 - angle;
end
```

## Klasse `Arc` als Pedant zum Bogenintervall $b_{ij}$

Zur Speicherung der Bogensegmente und deren genaue Lage auf dem Kreisbogen mittels Intervallen $b_{ij}$ (Definition 4.8) wird die Klasse `Arc` definiert:

**Quelltext 4.5:** Vereinfachter Code zur Klasse `Arc`

```
classdef Arc
       properties
              startPoint
              endPoint

       methods
              function obj = Arc(startPoint, endPoint) % Konstruktor
                     obj.startPoint = startPoint;
                     obj.endPoint = endPoint;
end
```

Für die Objektinitiierung werden lediglich die Parameter `startPoint` und `endPoint` benötigt, welche den Intervallgrenzen $\gamma_{ij}^{(1)}$ und $\gamma_{ij}^{(2)}$ aus Definition 4.8 entsprechen. Diese kennzeichnen Start- und Endpunkt des nicht überdeckten Bogenstücks auf dem Kreis $C_i$ in Bezug auf Nachbarkreis $C_j$.

## 4.2 Pixelbasierte Umfangs- und Oberflächenberechnung

### 4.2.1 Vom Kantenbild zur Oberfläche

Alternativ zur geometrischen Berechnung kann die Umfangslinie $l$ und Mantelfläche $F$ auch mithilfe der detektierten Kantenpixel aus Kapitel 3.3 abgeschätzt werden. Es wird die Annahme getroffen, dass die Summe der durch den Canny Filter extrahierten Pixel der Umfangslinie $l_{\text{px}}$ einer Kreiskonfiguration $C_1, C_2, ..., C_n$ (siehe Abb. 4.2) entspricht. In diesem Fall lässt sich die Mantelfläche analog zur geometrischen Berechnung direkt mit $F_{\text{px}} = l_{\text{px}} \cdot h$ angeben.

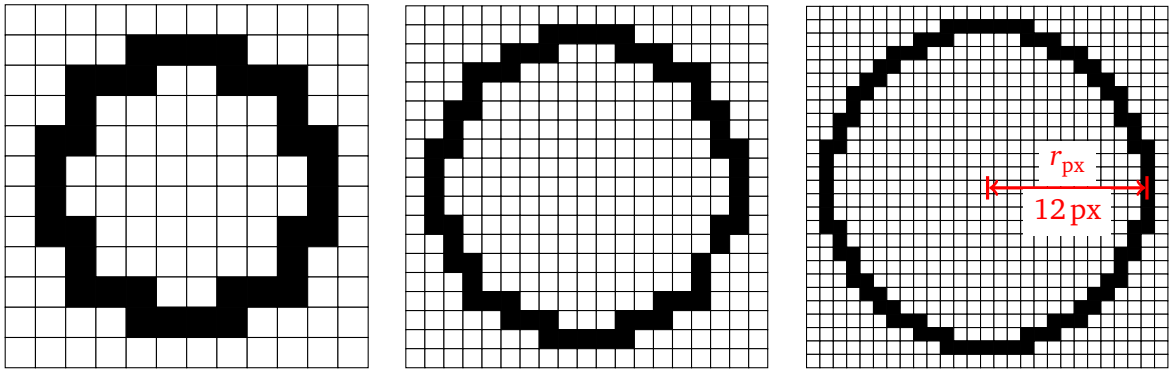
Diese Berechnungsmethode basiert auf der Annahme, dass die geometrische Umfangslinie $l_{\text{px}}$ durch die Summe der detektierten Kantenpixel $L_{\text{px}}$ am besten approximiert werden. Aus diesem Grund ist es besonders essentiell, dass zur Vermeidung von falsch positiv detektierten Kantenpixeln vor der Kantendetektion das Bild mit einem Glättungsfilter mathematisch gefaltet wird.

#### Definition von Rastergittergröße $n$ und Radius $r_{\text{px}}$ einer Kreisrasterung

Die Kreise und Bogenstücke, die im Rahmen dieser Arbeit untersucht werden, liegen nach der Kantendetektion ausschließlich als gerasterte Konturen vor. Diese Konturen haben bedingt durch den Canny Algorithmus stets die konstante Dicke von einem Pixel. Abhängig von der Größe der aufgenommenen Strukturen und der Auflösung der Aufnahme ergibt sich eine Rasterung der Größe $n \times n$. Diese Größe entspricht gerade der Pixelanzahl, mit der genau ein voller Kreis der Aufnahme gerastert werden kann. Für die nachfolgenden Abschnitte werden sämtlichen Längen und Maße in der Einheit Pixel (px) angegeben, was genau der Breite eines Pixels entspricht.

Um die Maße des ursprünglichen Kreises anhand der Rasterung rekonstruieren zu können, wird ein Zusammenhang zwischen der Pixelposition und des darunterliegenden Kreises hergestellt. Sind die Pixel nicht unendlich klein, muss festgelegt werden, welche Pixel den tatsächlichen Kreis abbilden. Dazu wird jedes Pixel, das vom geometrischen Kreis berührt wird, zur Kreisrasterung verwendet und entsprechend schwarz eingefärbt. Es wird implizit die Annahme getroffen, dass jedes Bogenstück auf solche Weise gerastert wird, dass die resultierenden Pixel zentral auf dem Kreisbogen liegen. Entsprechend ist der Radius $r_{\text{px}}$ des zugrundeliegenden Kreises über den Mittelpunkt des Gitters und das Zentrum des äußersten Pixels (siehe Start- und Endpunkt von $r_{\text{px}}$ in Abb. 4.6c) definiert.

Es gilt: Eine Rasterung von $n \times n$ bildet einen Kreis ab, der einen Radius $r_{\text{px}}$ nach Gleichung 4.14 besitzt.




**(a)** $10 \times 10$ **(b)** $17 \times 17$ **(c)** $25 \times 25$

**Abbildung 4.6.:** Gerasterte Kreise auf unterschiedlichen Gittergrößen

$$r_{\text{px}} := \frac{n-1}{2} \tag{4.14}$$

Da ein Bild aus einer endlichen Anzahl Pixeln besteht, eine geometrische Figur aber unendlich viele Punkte umfasst, ist eine direkte, fehlerfreie Ableitung der Umfangslinie aus der Kantenpixelanzahl nicht möglich. Als Veranschaulichung dieses Fehlers soll das folgende konstruierte Beispiel dienen.

#### Beispiel 1: Konstruierte Rasterung eines idealen Kreises auf einer Gittergröße von $25 \times 25$

Liegt eine Kreisrasterung[1] wie in Abb. 4.6c vor, so besitzt der zugehörige Kreis nach Gleichung 4.14 einen Radius $r_{\text{px}}$ von 12 px. Mithilfe dieser Information wird folgender Kreisumfang $l_{\text{px}}$ unabhängig von der Rasterung erwartet:

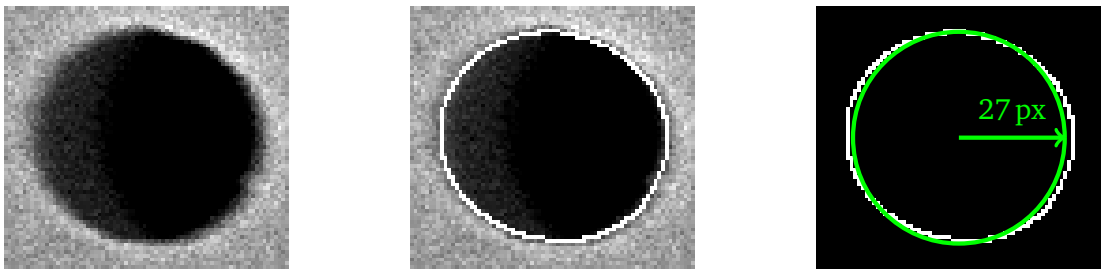
$$l_{\text{px}} = 2\pi r_{\text{px}} = 2\pi 12\,\text{px} = 75{,}4\,\text{px} \tag{4.15}$$

Wird allerdings die tatsächliche Anzahl der Pixel aus der Umfangslinie in Abb. 4.6c gezählt, ergibt sich eine Summe von $L_{\text{px}} = 96\,\text{px}$, was in diesem Fall einer relativen Abweichung von 27,3 % entspricht.

#### Beispiel 2: Rasterung eines Lochs aus einer REM-Aufnahme nach einer Kantenfilterung

In Abb. 4.7a ist ein Ausschnitt aus einer Elektronenmikroskop-Aufnahme (Abb. B.1, Loch 1) zu sehen, auf welchem ein einzelnes Loch der aufgenommenen Templatfolie abgebildet ist. Nach anschließender Kantenfilterung liegt lediglich eine Kontur (Abb. 4.7b, weiße Linie) vor, welche der Umfangslinie des Loches entspricht.

[1] Bresenham-Algorithmus [12] mit erhöhter Pixelstärke in den Kurven, für eine bessere Annäherung an das Ergebnis eines Canny Filters




**(a)** Ausschnitt aus Aufnahme

**(b)** Detektierte Umfangslinie (weiße Linie)

**(c)** Einzeichnung eines passenden Kreises mitsamt zugehörigem Radius

**Abbildung 4.7.:** Von der Aufnahme zur Bestimmung der Umfangslinienlänge anhand der Kantenpixel

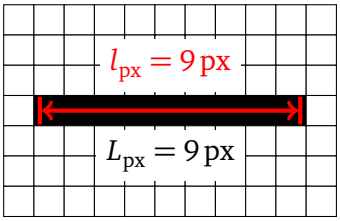
Mithilfe der Hough-Transformation kann die Position und der Radius eines in diese Kontur passenden Kreises bestimmt werden. Speziell für die Kontur in Abb. 4.7b ergibt sich ein Kreis mit dem Radius $r_{\mathrm{H,px}} = 27\,\mathrm{px}$, welcher in Abb. 4.7c als dicke Linie über der Kontur liegt. Werden alle Konturenpixel zusammengezählt, ergibt sich eine Summe von $L_{\mathrm{px}} = 216\,\mathrm{px}$. Allerdings entspricht die Umfangslänge eines Kreises, der einen Radius von 27 px besitzt, lediglich der Länge $l_{\mathrm{px}} = 2\pi 27\,\mathrm{px} \approx 167\,\mathrm{px}$, was einer relativen Abweichung von 27,1 % entspricht. Auch an dieser Stelle wird die Länge der Umfangslinie mithilfe der Kantenpixelzählung deutlich überschätzt.
Zusätzlich kann auch das Wissen um den Durchmesser der Löcher genutzt werden, um das Abschätzverfahren auf Tauglichkeit zu überprüfen. In Abschnitt 2.3 wurde der Begriff des Kalibrierungsfaktors eingeführt, mit dessen Hilfe die tatsächliche Länge in Mikrometern angegeben werden kann. Der in Abb. 4.7a zu sehende Ausschnitt ist einer REM-Aufnahme entnommen, welche mit 25.000-facher Vergrößerung eine Templatfolie zeigt. Diese Templatfolie ist mit Löchern perforiert, die einen normierten Durchmesser von 400 nm besitzen. Aus diesen Informationen kann der zu erwartende Radius $r_{\mathrm{H,px}}$ direkt mit 28,4 px angegeben werden[2]. Die daraus abgeleitete Umfangslänge liegt somit bei rund 178 px, wodurch sich der relative Fehler der Abschätzung durch Pixelzählung auf 21 % verringert.

Im nächsten Abschnitt werden die Gründe für diesen Fehler und Möglichkeiten der Kompensation vorgestellt.

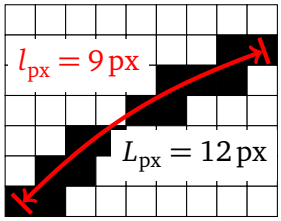
### 4.2.2 Fehler durch Kreisrasterung

Die Abweichung der Pixelzählung ist in der Diskretisierung des Kreisbogens begründet. Gerasterte horizontale oder vertikale Linien entsprechen in ihrer Abmessung exakt der zugrundeliegenden, geometrischen Form (Abb. 4.8a). Kurvige und diagonal über das Gitter verlaufende Linien werden dagegen durch treppenartig angeordnete Pixel nur angenähert (Abb. 4.8b) und benötigen deshalb insgesamt eine höhere Anzahl an Pixeln.

[2] Der Maßstabsbalken im Bild ist auf 284 px pro 2 $\mu$m normiert. Erwarteter Radius der Löcher beträgt 0,2 $\mu$m, was anhand des Maßstabes 28,4 px entspricht.




**(a)** Hier entspricht die Länge der Pixel genau der zugrundeliegenden Linie (1 Pixel ≡ 1 px)




**(b)** Dieses kurvige Bogenstück ist ein Ausschnitt der Kontur aus Abb. 4.7b (1 Pixel ≡ 0,7 px)

**Abbildung 4.8.:** Vergleich zwischen Rasterungen einer horizontalen Linie und einer diagonal verlaufenden Linie. Die Rasterung in (b) benötigt signifikant mehr Pixel.

Finden sich in dem zu analysierenden Bild eine hohe Anzahl an kurvigen Formen, so kann ohne Berücksichtigung des Diskretisierungsfehlers nicht ohne Weiteres die Kantenpixelanzahl als tatsächliche Länge aller Kurvenstücke bestimmt werden. Um die Auswirkung des Diskretisierungsfehlers abschätzen und kompensieren zu können, wird in Tabelle 4.1 eine Vielzahl von unterschiedlichen Kreisen der gleichen Aufnahme (Abb. B.1) auf Anzahl der Kantenpixel und den durch die Abschätzung verursachten Fehler untersucht.
Beachte: Die einzelnen Löcher werden in den Tabellen 4.1, 4.2 und 4.3 über den Identifier „<Abbildungsnummer>.<Lochnummer>“ referenziert. Die Abbildungen sind dem Anhang beigefügt.

Die beiden Fehler-Größen $f_{\text{H}}$ und $f_{\text{D}}$ bilden den prozentualen Fehler der Schätzmethode bezüglich der geometrischen Umfangslinienlängen $l_{\text{H,px}}$ und $l_{\text{D,px}}$, welche einerseits über die Hough-Transformation und andererseits über die Angaben zum Lochdurchmesser aus dem Datenblatt bestimmt werden.

$$f_H = \frac{L_{\text{px}} - l_{\text{H,px}}}{l_{\text{H,px}}} 100\,\% \qquad f_D = \frac{L_{\text{px}} - l_{\text{D,px}}}{l_{\text{D,px}}} 100\,\% \tag{4.16}$$

Weichen die Ergebnisse dieser Methode systematisch ab, können diese mithilfe eines Anpassungsfaktors effektiv kompensiert werden und zu einem befriedigenden Ergebnis führen. Die Frage, ob ein systematischer Fehler vorliegt, kann mithilfe der empirischen Standardabweichung $s_x$ und dem Varianzkoeffizienten $v_s$ (Definition 4.17) als Streuungsmaß beantwortet werden.

$$v_s = \frac{s_x}{\overline{x}} \tag{4.17}$$

Systematische Abweichungen erzeugen eine Verschiebung nach einer Seite, also eine Tendenz stets zu hohen oder zu niedrigen Messwerten [Quellenverweis zu betreffendem Wiki-Artikel]. In Tabellen 4.1, 4.2 und 4.3 wird aus den Fehlern direkt ersichtlich, dass hier eine eben solche einseitige Tendenz zu hohen Abweichungen vorliegt. Um allerdings den Begriff einer systematischen Abweichung enger zu fassen, wird die innere Genauigkeit dieser Abweichung mithilfe des Varianzkoeffizienten $v_s$ näher untersucht. Nach [Quellenverweis auf die englische Wiki-Seite zu Varianzkoeffizient] gelten Verteilungen als gering gestreut, wenn deren Varianzkoeffizient unter 100 % liegt und als hoch gestreut, wenn diese einen Varianzkoeffizienten von über 100 % aufweisen.

**Tabelle 4.1.:** Ergebnisse der Untersuchung einzelner Löcher aus Abb. B.1 auf Kantenpixelanzahl und daraus resultierende Abschätzungsfehler in Bezug auf den ermittelten Radius $r_{\mathrm{H,px}}$ als auch auf den im Vorfeld bekannten Radius $r_{\mathrm{D,px}} = 28{,}4\,\mathrm{px}$. Ermittelt wird $r_{\mathrm{H,px}}$ mithilfe der Hough Transformation. Entsprechend ist $f_{\mathrm{H}}$ der prozentuale Fehler in Bezug auf $r_{\mathrm{H,px}}$ und $f_{\mathrm{D}}$ der prozentuale Fehler in Bezug auf $r_{\mathrm{D,px}}$. Weiter bezeichnet $\overline{x}$ den Mittelwert, $s_x$ die emp. Standardabweichung und $v_s$ den emp. Variationskoeffizienten der Messreihe. Letztere sollen die Beurteilung hinsichtlich eines systematischen Fehlers vereinfachen.

| | | Hough-T. | | |
|---|---|---|---|---|
| **Loch** | $L_{\mathbf{px}}$ | $r_{\mathrm{H,px}}$ | $f_H$ | $f_D$ |
| B1.1 | 202 | 30 | 7,2 | 13,2 |
| B1.2 | 209 | 27 | 23,2 | 17,1 |
| B1.3 | 198 | 26 | 21,2 | 11,0 |
| B1.4 | 211 | 30 | 11,9 | 18,2 |
| B1.5 | 206 | 27 | 21,4 | 15,4 |
| B1.6 | 184 | 25 | 17,1 | 3,1 |
| B1.7 | 204 | 26 | 24,9 | 14,3 |
| B1.8 | 209 | 30 | 10,9 | 17,1 |
| $\overline{x}$ | 203 | 27.6 | 17,2 | 13,7 |
| $s_x$ | 8,7 | 2,1 | 6,5 | 4,9 |
| $v_s$ | | | 37,8 | 35,7 |

Im Rahmen dieser Untersuchung wird willkührlich festgelegt, dass eine systematische Abweichung genau dann vorliegt, wenn der Varianzkoeffizient der Messreihe bei unter 100 % liegt, also eine hohe innere Genauigkeit aufweist. Dies ist in den Untersuchungen für die Aufnahmen B.2 und B.3 der Fall, so dass davon ausgegangen werden kann, dass hier eine systematische Abweichung vorliegt. Weiter lässt sich aus den Ergebnissen schlussfolgern, dass sich die Tendenz zur systematischen Abweichung bei Aufnahmen von Löchern aus großen Rastergittern entwickelt.

### Unterschiedliche Kreisdurchmesser

Um festzustellen, ob es einen Zusammenhang zwischen Rastergitter-Größe, resultierenden Fehlern und deren Streuung gibt, werden zusätzlich Löcher aus Aufnahmen unterschiedlicher Zoom-Stufen untersucht (siehe Abb. B.1 und Abb. B.3). Die hieraus entstandenen Ergebnisse in Tabellen 4.2 und 4.3 sollen Aufschluss darüber geben, ob Abschätzungsfehler durch die Wahl des Zoomlevels beeinflussbar sind.

Es kann festgestellt weren, dass die prozentualen Messabweichungen $f_H$ und $f_D$ systematisch größer werden, je größer die zu untersuchenden Löcher auf der Aufnahme abgebildet sind. Gleichzeitig stabilisieren sich sowohl Streuung als auch der Varianzkoeffizient mit steigendem Abbildungsmaßstab. Umgekehrt wird der Varianzkoeffizient für Aufnahmen kleinen Maßstabs beliebig groß. Er liegt in diesem Fall bei über 100 % und macht damit eine sinnvolle Fehlerkompensation nahezu unmöglich.

**Tabelle 4.2.:** Untersuchsreihe zu Löchern im kleinen Maßstab in Bezug auf Abb. B.2. Der bekannte Lochradius beträgt hier $r_{\mathrm{D,px}} = 16{,}8\,\mathrm{px}$.

| | | Hough-T. | | |
|---|---|---|---|---|
| **Loch** | $L_{\mathbf{px}}$ | $r_{\mathrm{H,px}}$ | $f_H$ | $f_D$ |
| B2.1 | 120 | 18 | 6,1 | 13,7 |
| B2.2 | 115 | 14 | 30,7 | 8,9 |
| B2.3 | 114 | 18 | 0,8 | 8,0 |
| B2.4 | 104 | 17 | −2,6 | −1,5 |
| B2.5 | 104 | 12 | 37,9 | −1,5 |
| B2.6 | 119 | 19 | −0,3 | 12,7 |
| B2.7 | 103 | 13 | 26,1 | −2,4 |
| B2.8 | 119 | 14 | 35,3 | 12,7 |
| $\overline{x}$ | 112 | 15,6 | 16,7 | 6,3 |
| $s_x$ | 7,4 | 2,7 | 17,4 | 7,0 |
| $v_s$ | | | 103,7 | 110,6 |

**Tabelle 4.3.:** Untersuchsreihe zu Löchern im großen Maßstab in Bezug auf Abb. B.3. Der bekannte Lochradius beträgt hier $r_{\mathrm{D,px}} = 74{,}4\,\mathrm{px}$.

| | | Hough-T. | | |
|---|---|---|---|---|
| **Loch** | $L_{\mathbf{px}}$ | $r_{\mathrm{H,px}}$ | $f_H$ | $f_D$ |
| B3.1 | 561 | 77 | 16,0 | 20,0 |
| B3.2 | 553 | 77 | 14,3 | 18,3 |
| B3.3 | 597 | 79 | 20,3 | 27,7 |
| B3.4 | 547 | 78 | 11,6 | 17,0 |
| B3.5 | 565 | 77 | 16,8 | 20,9 |
| B3.6 | 577 | 78 | 17,7 | 23,4 |
| B3.7 | 554 | 76 | 16,0 | 18,5 |
| B3.8 | 575 | 78 | 7,3 | 23,0 |
| $\overline{x}$ | 566 | 77,5 | 15,0 | 21,1 |
| $s_x$ | 16,3 | 0,9 | 4,0 | 3,5 |
| $v_s$ | | | 26,7 | 16,6 |

### 4.2.3 Fehlerkompensation

Im vorherigen Abschnitt wurde gezeigt, dass die systematische Abweichung mit der Abbildungsmaßstab der Aufnahme der korreliert. Um diese Abweichung kompensieren zu können, wird ein Korrekturfaktor $k_D$ eingeführt, der direkt über den Mittelwert des Fehlers $f_{\mathrm{D}}$ bestimmt wird.

$$k_D := \frac{1}{1 + \frac{\overline{f_D}}{100\,\%}} \tag{4.18}$$

Auf diese Weise kann die Summe der gezählten Kantenpixel $L_{\mathrm{px}}$ direkt mit dem Korrekturfaktor $k_D$ nach unten korrigiert werden, um den neuen Schätzwert $\hat{L}_{\mathrm{px}}$ für die Umfangslinienlänge $l_{\mathrm{px}}$ zu erhalten.

$$\hat{L}_{\mathrm{px}} = k_D L_{\mathrm{px}} \tag{4.19}$$

### 4.2.4 Vor- und Nachteile

Ein wesentlicher Vorteil der Methode der Kantenpixelzählung ist die Ausführungsgeschwindigkeit. Zeitmessungen hierzu folgen in Tabellen 5.1, 5.2 und 5.3 in Kapitel 5.
Ein weiterer Vorteil dieses Verfahrens ist die Berücksichtigung von Objekten am Rande eines Bildes. Bei der Hough-Transformation dagegen werden Löcher, deren Mittelpunkt nicht mehr im Bildbereich detektierbar sind, vernachlässigt, wodurch ein nicht vernachlässigbarer Fehler entsteht.

Nachteilig wirkt sich der (systematische) Schätzfehler durch Verwendung einer zum Messergebnis führenden Beziehung zwischen Größen, die der tatsächlichen Verknüpfung nicht entspricht. Explizit geht es dabei um die Verknüpfung $L_{\mathrm{px}} \equiv l_{\mathrm{px}}$, die in Wirklichkeit nicht gegeben ist. Da bei ausreichend großen Strukturen in der Aufnahme eine systematische Abweichung dieser Verknüpfung angenommen werden kann, wurde in Gleichung 4.18 der Korrekturfaktor $k_D$ definiert, um über eine neue Verknüpfung $k_D L_{\mathrm{px}} \equiv l_{\mathrm{px}}$ den Schätzfehler der Umfangslinienlänge zu kompensieren.
Ein weiterer Nachteil der Kantendetektion ist die Empfindlichkeit gegenüber Rauschen des Bildsensors und physischen Artefakten auf der zu untersuchenden Oberfläche. Unebenmäßigkeiten auf der Substratgrundfläche werden ebenfalls als Kantenpixel extrahiert und fälschlicherweise zur Umfangslinienlänge hinzugenommen.

# 5 Ergebnisse und Vergleich

Im Verlauf dieser Arbeit wurden zwei Möglichkeiten der Oberflächenerfassung vorgestellt. Welche für den vorliegenden Anwendungsfall die geeignetere ist, wird in diesem Kapitel untersucht.

## 5.1 Festlegung der Bewertungskriterien

Folgende Punkte sollen als Vergleichskriterien dienen:

- **Robustheit**
  Das Verfahren ist robust gegenüber verschiedenen Bildaufnahmeeinstellungen, Auflösung und Größe der zu detektierenden Kreise. Zusätzlich werden diese Kriterien nicht nur unter einer einzigen Bildaufnahme verglichen, sondern auch mit unterschiedlichen Aufnahmeparametern gegenübergestellt.
  Mithilfe der Einstellungen, welche direkt bei der Aufnahme am REM ausgewählt werden können, wurden für die Gegenüberstellung folgende Werte in drei äquidistanten Stufen für den Vergleich herangezogen:
  - **Helligkeit**
  - **Kontrast**
  - **Auflösung**
- **Korrekte Anzahl detektierter Kreise**
  Das Maß an korrekt detektierten Kreisen gegenüber der tatsächlich vorliegenden Menge. Hierbei wird die Anzahl ausgelassener Kreise (unter der Spalte „M“ in den folgenden Vergleichstabellen) und die Anzahl von falsch positiven Detektionen (Spalte „FP“) gegenübergestellt.
- **Zeitperformance**
  Die benötigte Zeit zur Berechnung der Mantelfläche $F$ in Sekunden.

Als Referenzwert für die Gegenüberstellung wird aufgrund der sehr präzisen Ergebnisse in den Versuchsreihen das Verfahren der Hough-Transformation mit der gekoppelten geometrischen Umfangslinienberechnung gewählt. Damit soll vor allem überprüft werden, ob das Maß an Zeiteinsparung die ungenauere Oberflächenberechnung durch den pixelbasierten Ansatz kompensiert werden kann[1].

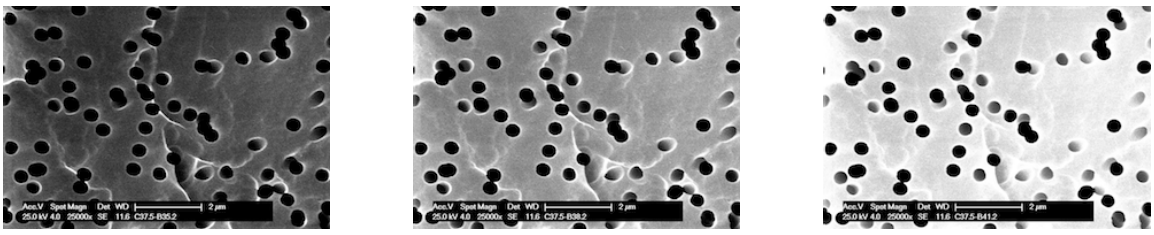
[1] Allerdings spielt im Rahmen der Aufgabenstellung dieser Arbeit der Zeitkostenpunkt eine untergeordnete Rolle, da nur eine relativ begrenzte Anzahl an Bildern zur Verfügung steht.

## 5.2 Ergebnisse

Nachfolgend werden sämtliche Messergebnisse hinsichtlich berechneter Mantelfläche $F$ beider in Kapitel 4 vorgestellten Schätzmethoden aufgeführt. Alle Durchläufe wurden dabei im *single-thread*-Modus direkt in Matlab ausgeführt. Die Zeitperformance kann sich von Rechnerarchitektur zu Rechnerarchitektur unterscheiden und soll lediglich einen groben Anhaltspunkt für die benötigte Zeit geben.

### 5.2.1 Variation der Helligkeit

Für die folgende Gegenüberstellung werden die Verfahren auf Robustheit im Bezug auf Helligkeitsveränderungen untersucht. Als Grundlage dienen die Aufnahmen in Abb. 5.1.



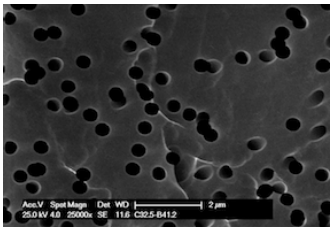
**(a)** Niedrige Helligkeit (B35.2) **(b)** Mittlere Helligkeit (B38.2) **(c)** Hohe Helligkeit (B41.2)

**Abbildung 5.1.:** Von der Aufnahme zur Bestimmung der Umfangslinienlänge anhand der Kantenpixel

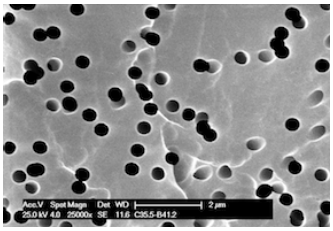
**Tabelle 5.1.:** Ergebnisse hinsichtlich verschiedener Helligkeitseinstellungen. Hierbei bezeichnet $G_A$ den generisch bestimmten Sensitivitätswert (s. Abschnitt 3.2.1) und $G_H$ denjenigen, der manuell durch Ausprobieren gewählt wurde. Weiter bezeichnet $F$ die berechnete Mantelfläche, welche der Innenseite der Poren der Template entsprechen. M (missed) steht für die Anzahl nicht detektierter Löcher und FP (false positive) für die Anzahl von Falschdetektionen. Als $f_{\text{geo}}$ wird der prozentuale Fehler der berechneten Mantelfläche $F_{\text{px}}$ der pixelbasierten Methode im Bezug zur berechneten Fläche $F$ der geometrischen Methode bezeichnet.

| | **Geom. Ansatz** | | | | | **Pixelbasiert** | | |
|---|---|---|---|---|---|---|---|---|
| **Helligkeit** | Sensitivität | $F$ ($\mu$m$^2$) | M | FP | Zeit (s) | $F_{\text{px}}$ ($\mu$m$^2$) | Zeit (s) | $f_{\text{geo}}$ |
| B35.2 | $G_A = 0.940$ | 995 | 6 | 7 | 3,01 | | | |
| | $G_H = 0.940$ | 995 | 6 | 7 | 3,01 | 1033 | $\ll 1$ | 3,8 |
| B38.2 | $G_A = 0.915$ | 788 | 8 | 0 | 3,06 | | | |
| | $G_H = 0.930$ | 882 | 1 | 0 | 3,22 | 1228 | $\ll 1$ | 39,2 |
| B41.2 | $G_A = 0.880$ | 624 | 19 | 0 | 2,42 | | | |
| | $G_H = 0.925$ | 880 | 2 | 0 | 2,90 | 1422 | $\ll 1$ | 61,6 |

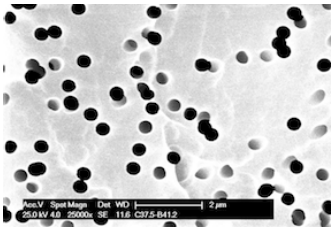
### 5.2.2 Variation des Kontrasts



**(a)** Niedriger Kontrast (C32.5)



**(b)** Mittlerer Kontrast (C35.5)



**(c)** Hoher Kontrast (C37.5)

**Abbildung 5.2.:** Von der Aufnahme zur Bestimmung der Umfangslinienlänge anhand der Kantenpixel

**Tabelle 5.2.:** Ergebnisse hinsichtlich verschiedener Kontrasteinstellungen

| | Geom. Ansatz | | | | | Pixelbasiert | | |
|---|---|---|---|---|---|---|---|---|
| **Kontrast** | Sensitivität | $F$ ($\mu$m$^2$) | M | FP | Zeit (s) | $F_{\text{px}}$ ($\mu$m$^2$) | Zeit (s) | $f_{\text{geo}}$ |
| C32.5 | $G_A = 0.925$ | 859 | 6 | 1 | 3,12 | | | |
| | $G_H = 0.925$ | 859 | 6 | 1 | 3,12 | 1465 | $\ll 1$ | 70,5 |
| C35.5 | $G_A = 0.935$ | 925 | 0 | 0 | 3,06 | | | |
| | $G_H = 0.935$ | 925 | 0 | 0 | 3,42 | 1366 | $\ll 1$ | 46,1 |
| C37.5 | $G_A = 0.880$ | 624 | 19 | 0 | 2,42 | | | |
| | $G_H = 0.925$ | 880 | 2 | 0 | 2,90 | 1422 | $\ll 1$ | 61,6 |

### 5.2.3 Auflösung

**Tabelle 5.3.:** Ergebnisse hinsichtlich verschiedener Auflösungen der Aufnahme in Abb. 5.2b

| | Geom. Ansatz | | | | | Pixelbasiert | | |
|---|---|---|---|---|---|---|---|---|
| **Auflösung** | Sensitivität | $F$ ($\mu$m$^2$) | M | FP | Zeit (s) | $F_{\text{px}}$ ($\mu$m$^2$) | Zeit (s) | $f_{\text{geo}}$ |
| C35.5(.5×) | $G_A = 0.940$ | 888 | 2 | 0 | 0,67 | | | |
| | $G_H = 0.940$ | 888 | 2 | 0 | 0,67 | 1025 | $\ll 1$ | 15,4 |
| C35.5(1×) | $G_A = 0.935$ | 925 | 0 | 0 | 3,06 | | | |
| | $G_H = 0.935$ | 925 | 0 | 0 | 3,42 | 1366 | $\ll 1$ | 46,1 |
| C35.5(2×) | – | – | – | – | – | | | |
| | $G_H = 0.945$ | 894 | 1 | 0 | 21,25 | 1241 | $\ll 1$ | 38,8 |

## 5.3 Fazit

Wie erwartet hat im besonderen Maße die geometrische Methode die besten Ergebnisse hinsichtlich einer genauen Flächenberechnung gezeigt. Die pixelbasierte Methode konnte

bestenfalls bei guten Helligkeits- und Kontrastwerten mit einem kleinen Fehler eine gute Näherung geben. In allen anderen Fällen ist die Abweichung durch bloße Kantenlängenzählung mit durchgeführter Kompensation zu hoch, um praktischen Einsatz finden zu können. Die Zeiteinsparung ist bei den auftretenden kurzen Durchlaufzeiten gegenüber dem geometrischen Pendant vernachlässigbar.

Wünschenswert war die Übereinstimmung der Sensitivitätswerte $G_A$ und $G_H$, da dies mit einer erfolgreichen, vollautomatischen Parameterberechnung gleichzusetzen ist. $G_H$ kann damit auch stets als Referenzwert für den generisch ermittelten Arbeitspunkt $G_A$ angesehen werden.
In dieser Hinsicht war die Qualität des generischen Ansatzes für eine Vielzahl von Bildern sehr zufriedenstellend. Eine relativ starke Abweichung trat nur bei sehr hellen Bildern auf (siehe Abb. 5.1c und Abb. 5.2c sowie Tabelle 5.1, letzter Eintrag).

Das Bild mit den günstigsten Aufnahmeparametern war jenes aus Abbildung 5.2b. Bei dieser Aufnahme wurden sowohl alle sichtbaren Kreise als auch keine falsch positiven Ergebnisse detektiert. Zusammenfassend kann die Aussage getroffen werden, dass Aufnahmen für eine möglichst genaue Flächenberechnung eine mittlere Helligkeit und einen mittleren Kontrast aufweisen sollen.

Auffällig in diesen Messreihen war die starke Korrelation zwischen Bildauflösung und Ausführungszeit. Eine einzelne Kreisdetektion bei doppelter Auflösung hat die Ausführungszeit um den Faktor 7 erhöht. Aus diesem Grund war eine Zeitmessung mit generisch bestimmtem Sensitivitätswert $G_A$ zeitlich nicht möglich, da der Zeitaufwand der Iterationen über den Arbeitsbereich die in der Praxis verfügbare Zeit deutlich überstieg. Auch hier ist eine mittlere Auflösung von etwa $1000 \times 2000$ vorzuziehen. Auflösungen unter diesem Ankerpunkt erzeugen eine größere Abweichung als Auflösungen darüber. Folglich ist eine niedrige Auflösung stets zu vermeiden.

Insgesamt kann die Arbeit mit dem Urteil geschlossen werden, dass eine hinreichend genaue geometrische Oberflächenbestimmung von REM-Aufnahmen mit dem Ansatz der Hough-Transformation und der trigonometrischen Berechnung aller Umfangslinien durchführbar ist.

# 6 Weiterführende Arbeit

## 6.1 Allgemeine Hough-Transformation

Eine weitere Möglichkeit zur Lösung des Detektionsproblems über den geometrischen Ansatz ist der Einsatz der generalisierten Hough-Transformation [13]. Hier würde sich beispielsweise die Parametrisierung einer halben Ellipse anbieten, da sich innerhalb der Versuchsreihen immer wieder gezeigt hatte, dass durch eine Kantenextraktion viele halbe Ellipsen hervortreten. Diese entstanden meist durch seitliche Ionenbeschüsse, so dass bei der Aufnahme einseitige Verschmierungen durch unzureichenden Kontrast zwischen Lochinnenseite und Lochumriss entstanden. In einem mehrstufigen Prozess könnte man auf diese Weise die Genauigkeit der detektierten Lochpositionen unter einem kleineren Schwellenwert realisieren.

## 6.2 Hinzunahme weiterer quantifizierbarer Größen

Eine Verbesserung des Kreisdetektions-Algorithmus kann durch Hinzunahme weiterer, extrahierbarer Informationen noch verbessert werden. Beispielsweise kann der Ausgabeparameter `metric` der Funktion `imfindcircles` angeführt werden. Dieser sortiert die gefundenen Kreise in einer absteigenden Reihenfolge anhand ihrer Kantenstärke. Aus diesen Größen kann der Zusammenhang zu nicht orthogonal geschossenen Löchern hergestellt werden. Löcher, die durch einen seitlichen Einschuss entstehen, erscheinen beim Detektionsvorgang als unfertige Bogensegmente und landen damit im `metric`-Array sehr weit unten. Das ist deshalb wichtig, weil ein nicht orthogonal erzeugtes Loch einen verzogenen Umriss aus Perspektive des Mikroskopobjektivs besitzt und das damit sowohl die geometrische als auch die pixelbasierte Umfangs- und Oberflächenberechnung erschwert.

## 6.3 Parallelisierung des Algorithmus

Die vorliegende Implementierung der Arbeitspunktsuche ist so geschrieben, dass jede Iteration entlang des Sensitivitätsbereich von einem einzigen Prozess (Thread) berechnet wird. Da die Kreisdetektion für unterschiedliche Sensitivitätswerte jedoch vollkommen unabhängig voneinander ablaufen, können theoretisch beliebig viele Einzeliterationen in unterschiedlichen Prozessen abgearbeitet und damit insgesamt deutlich beschleunigt werden.

## 6.4 Machine Learning

Das Fachgebiet des Maschinellen Lernens bietet die Möglichkeit bereits detektierte Strukturen auf ihre korrekte Erkennung zu überprüfen. Dazu könnten mithilfe eines Trainingssets, welches unterschiedliche Ausschnitte von Löchern einer Templatfolie beinhaltet, ein Klassifizierungsmodell erstellt werden. Dieses ist dann in der Lage eine Wahrscheinlichkeit anzugeben, ob die untersuchte Struktur einem Kreis entspricht. Diese Vorgehensweise setzt jedoch die bereits detektierten Kreise voraus und ist nicht in der Lage von sich aus die Positionen

der Kreise zu erfassen. Dementsprechend hat die Klassifizierung einer eher unterstützende Aufgabe bei der Verifizierung der Ergebnisse. Nachteile treten in Form von hoher Sensibilität für Variationen in Lichtverhältnissen, Rotation und Skalierung des Kreises auf. Deshalb müssen die Bedingungen unter denen die Bilder aufgenommen werden, sehr präzise kontrolliert werden.

## 7 Literaturverzeichnis


[1] HOUGH, P. V. C. *Method and means for recognizing complex patterns*. US Patent, 1962.

[2] CANNY, J. F. *A computational approach to edge detection*. S. 679-698, 1986.

[3] BURGER, W., BURGE, M. J. *Digitale Bildverarbeitung*. Springer-Verlag Berlin, 2006

[4] ATHERTON, T. J., KERBYSON, D. J. *Image and Vision Computing*. Volume 17, Number 11, 1999.

[5] BRUNELLI, R.. *Template Matching Techniques in Computer Vision: Theory and Practise*. Wiley, 2009.

[6] SPEISER, B.. *Cyclische Voltammetrie*. 1981.

[7] BRADSKI, G., KAEHLER, A.. *Learning OpenCV Computer Vision with the OpenCV Library*. O'Reilly, 2008.

[8] MATHWORKS.COM. *Image Processing Toolbox*. MATLAB and Simulink [Internet], verfügbar auf: https://de.mathworks.com/products/image.html.

[9] MATHWORKS.COM. *Detect and Measure Circular Objects in an Image*. MATLAB and Simulink [Internet], 2013, verfügbar auf: http://www.mathworks.com/help/images/examples/detect-and-measure-circular-objects-in-an-image.html.

[10] MARR, D.. *Vision*. Freeman, 1982.

[11] BAUMANN, R. *Geometrie: Winkelfunktionen, Trigonometrie*. S.76-77, Mentor, 1999.

[12] BRESENHAM, J. E. *A Linear Algorithm for Incremental Digital Display of Circular Arcs*. S.100-106, 1977.

[13] BALLARD, D. H., BROWN, C. M. *Computer Vision*. Prentice-Hall, 1982.

# A Code

## A.1 calculateSurfaceByDIP.m

**Quelltext A.1:** Verfeinerung der lokalen Minima

```
function [surface,sensitivity] = calculateSurfaceByDIP(varargin)
%CALCULATESURFACEBYDIP Calculate surface of cylindric objects
%   SURFACE = CALCULATESURFACEBYDIP(IMAGE) calculates the surface of cylindric
%   objects in the input image IMAGE. IMAGE can be a grayscale, RGB or
%   binary image. SURFACE is calculated by the position of found circles,
%   the default cylinder height of 24 micrometers and the default scale
%   factor of 1/142 micrometers per pixel.
%
%   [SURFACE, SENSITIVITY] = CALCULATESURFACEBYDIP(IMAGE) returns the
%   determined SENSITIVITY used for locating circle positions. Can be
%   useful for calibration of a series of images of the same record
%   parameters.
%
%   SURFACE = CALCULATESURFACEBYDIP(...,PARAM1,VAL1,PARAM2,VAL2,...)
%   calculates surface using name-value pairs to control aspects of the
%   Circular Hough Transform, scaling and visualization. Parameter names
%   can be abbreviated.
%
%   Parameters include:
%
%   'ObjectPolarity' - Specifies the polarity of the circular object with
%                   respect to the background. Available options are:
%
%                   'bright' : The object is brighter than the background.
%                   'dark'  : The object is darker than the background. (Default)
%
%   'Method' - Specifies the technique used for computing the operation
%             point. Available options are:
%
%             'BinarySearch' : Binary search to find the operation range.
%                             (Default)
%             'QuickEstimate' : Very quick method to find a mediocre
%                             operation range
%
%   'Sensitivity ' - Specifies the sensitivity factor in the range [0 1]
%                  for finding circles. A high sensitivity value leads
%                  to detecting more circles, including weak or
%                  partially obscured ones, at the risk of a higher
%                  false detection rate. Default value: Computed by the
%                  method which is specified by the 'Method' param.
%
%   'ScaleFactor' - Specifies how many pixels are needed to match one
%                  micrometer.
%                  Some values for specific zoom levels:
%                   8000x : 0.0219/1.4
%                  15000x : 0.012/1.4
%                  20000x : 0.0088/1.4
%                  25000x : 0.0071/1.4 (default)
%                  65000x : 0.0027/1.4
```

```
%                   yx : 175.2/y/1.4
%
%   'Height' - HEIGHT overrides the default HEIGHT of the cylinders.
%              Unit: micrometer.
%
%   'Visualization' - Specifies the visualization of found circles and arcs
%                   in the original image.
%
%                   'on' : Activate visualization
%                   'off' : Deactivate visualization (default)
%
%   Notes
%   -----
%   1.  Binary images (must be logical matrix) undergo additional pre-processing
%       to improve the accuracy of the result. RGB images are converted to
%       grayscale using RGB2GRAY before they are processed.
%   2.  Accuracy is limited for very small radius values, e.g. Rmin <= 5.
%   3.  The sensitivity estimation step for QuickSearch method is typically
%       faster than that of the BinarySearch method.
%   4.  Both Phase Coding and Two-Stage methods in IMFINDCIRCLES are limited
%       in their ability to detect concentric circles. The results for
%       concentric circles may vary based on the input image.
%   5.  IMFINDCIRCLES does not find circles with centers outside the image
%       domain.
%
%   Class Support
%   -------------
%   Input image A can be uint8, uint16, int16, double, single or logical,
%   and must be nonsparse. The output variables CENTERS, RADII, and METRIC
%   are of class double.

parsedInputs = parseInputs(varargin{:});

A = parsedInputs.Image;
method      = lower(parsedInputs.Method);
objPolarity = lower(parsedInputs.ObjectPolarity);
scaleFactor  = parsedInputs.ScaleFactor;
sensitivity = parsedInputs.Sensitivity;
height = parsedInputs.Height;
visual  = parsedInputs.Visualization;

%% CROP IMAGE
speedScale = 1;
A = imresize(A,1.4*speedScale);
A = A(1:floor(end*0.85),:);

%% READ SCALE BAR
radius = floor(37*speedScale);
% [radius,scaleFactor] = readScaleBar(A);

%% CALCULATE SENSITIVITY VALUE
if sensitivity == 0.0
    sensitivity = calculateSensitivityPoint(A, radius);
end

%% DETECT CIRCLES
[centers,radii] = imfindcircles(A,[radius-floor(radius*0.19) radius+floor(radius*0.19)] ...
,'ObjectPolarity',objPolarity,'Method','twostage','Sensitivity',sensitivity);
radius = mean(radii);
```

```
109
110 %% CALCULATE CIRCUMFERENCE OF ALL CIRCLES
111 circumference = calculateCircumference(centers,radius,scaleFactor);
112
113 %% CALCULATE SURFACE
114 surface = circumference * height;
115
116 %% VISUALIZATION?
117 if strcmp(visual,'on')
118     imshow(A)
119     radii = radius * ones(size(centers,1),1);
120     viscircles(centers, radii,'EdgeColor','b');
121     calculateCircumference(centers,radius,scaleFactor);
122
123     [x,y,~] = size(A);
124     surfacePlanar = x*y*scaleFactor*scaleFactor;
125     fprintf('Ebene␣Flaeche:␣%.2f␣Quadratmikrometer␣\n',surfacePlanar)
126     fprintf('Mantelflaeche:␣%.2f␣Quadratmikrometer␣\n',surface)
127     growth = surface/surfacePlanar + 1;
128     fprintf('Vergroesserung␣der␣Oberflaeche:␣%.2f␣%%␣\n',growth*100)
129 end
130 end
131
132 function parsedInputs = parseInputs(varargin)
133
134 % Validate number of input arguments
135 narginchk(1,Inf);
136
137 persistent parser;
138
139 % Wenn der Parser noch nicht initialisiert ist, initialisiere
140 if isempty(parser)
141     checkStringInput = @(x,name) validateattributes(x, ...
142         {'char','string'},{'scalartext'},mfilename,name);
143     parser = inputParser();
144     parser.addRequired('Image',@checkImage);
145     parser.addParameter('Method','BinarySearch',@(x) checkStringInput(x,'Method'));
146     parser.addParameter('ObjectPolarity','dark');
147     parser.addParameter('ScaleFactor',0.0071/1.4);
148     parser.addParameter('Sensitivity',0.0,@checkSensitivity);
149     parser.addParameter('Height',24);
150     parser.addParameter('Visualization', 'off', @(x) checkStringInput(x,'Visualization'));
151     % instead '0.85' use func like 'computeSensitivity'
152 end
153
154 % Parse input, replacing partial name matches with the canonical form.
155 if (nargin > 1) % If any name-value pairs are given
156     varargin(2:end) = images.internal.remapPartialParamNames({'Method', 'ObjectPolarity', ...
157         'ScaleFactor', 'Sensitivity', 'Height', 'Visualization'}, ...
158         varargin{2:end});
159 end
160
161 parser.parse(varargin{:});
162 parsedInputs = parser.Results;
163 parsedInputs.Method = checkMethod(parsedInputs.Method);
164 parsedInputs.Visualization = checkVisualization(parsedInputs.Visualization);
165
166     function tf = checkImage(A)
167         allowedImageTypes = {'uint8', 'uint16', 'double', 'logical', 'single', 'int16'};
```

```
        validateattributes(A,allowedImageTypes,{'nonempty',...
            'nonsparse','real'},mfilename,'A',1);
        N = ndims(A);
        if (isvector(A) || N > 3)
            error(message('images:imfindcircles:invalidInputImage'));
        elseif (N == 3)
            if (size(A,3) ~= 3)
                error(message('images:imfindcircles:invalidImageFormat'));
            end
        end
        tf = true;
    end

    function str = checkMethod(method)
        str = validatestring(method, {'BinarySearch','QuickSearch'}, ...
            mfilename, 'Method');
    end

    function tf = checkSensitivity(s)
        validateattributes(s,{'numeric'},{'nonempty','nonnan', ...
            'finite','scalar'},mfilename,'Sensitivity');
        if (s > 1 || s < 0)
            error(message('images:imfindcircles:outOfRangeSensitivity'));
        end
        tf = true;
    end

    function str = checkVisualization(v)
        str = validatestring(v, {'on','off'}, mfilename,'Visualization');
    end

end
```

## A.2 calculateSensitivityPoint.m

**Quelltext A.2:** Verfeinerung der lokalen Minima

```
function sensitivityPoint = calculateSensitivityPoint(image,radius)
%CALCULATESENSITIVITYPOINT Summary of this function goes here
%   Detailed explanation goes here

    %% ARBEITSBEREICH FUER DEN SENSITIVITAETSWERT ERMITTELN
    % 1) STARTWERT
    % Anfangswerte fuer den Sensitivitaetswert und die Schrittweite
    startPunkt = 0.5;
    schrittWeite = 0.25;

    % Genauigkeit der binaeren Suche einstellen
    epsilon = 0.01;

    % Beginne Zeitmessung
    tic

    % Bild einlesen
    %original = imread(image);

    % Binaere Suche starten
```

```
    while schrittWeite > epsilon
        foundCircles = imfindcircles(image,[radius-floor(radius*0.19) ...
         radius+floor(radius*0.19)],'ObjectPolarity','dark', ...
        'Method','twostage','Sensitivity',startPunkt);
        if ~isempty(foundCircles)
            startPunkt = startPunkt - schrittWeite;
        else
            startPunkt = startPunkt + schrittWeite;
        end

        schrittWeite = schrittWeite/2;
    end

    % Das Ergebnis auf zwei Dezimalstellen runden
    startPunkt = round(startPunkt,2) + 0.01;

    % 2) ENDWERT
    arrayCounter = 0;

    % Suche grob nach Kreisen im eingeschraenkten Sensitivitaetsbereich
    for i = startPunkt: 0.01: 0.99
        arrayCounter = arrayCounter + 1;
        [centers,~] = imfindcircles(image,[radius-floor(radius*0.19) ...
        radius+floor(radius*0.19)],'ObjectPolarity','dark', ...
        'Method','twostage','Sensitivity',i);
        if length(centers) < 500 % natuerliche, sinnvolle Maximalanzahl in einem Bild
            foundCircles(arrayCounter) = length(centers);
        else
            break;
        end
    end

    % Leite diskrete Funktion ab
    foundCirclesDiff = diff(foundCircles);

    % Bestimme Median der Menge
    medianValue = median(foundCirclesDiff);

    % Bestimme die Stelle, an der der Anstieg der Anzahl der gefundenen
    % Kreise den Median um 1000% uebertrifft
    indexOfSensitivity = find(foundCirclesDiff>medianValue*5, 1, 'first');
    endPunkt = startPunkt + 0.01* indexOfSensitivity;

    % Beende Zeitmessung
    toc

    %% SENSITIVITAETSWERT INNERHALB DES ARBEITSBEREICHS ERMITTELN
    % UEBER SENSITIVITAET ITERIEREN UND ANZAHL GEFUNDENER KREISE SPEICHERN
    foundCircles = zeros(1, 1000);
    arrayCounter = 1;
    for i=startPunkt:0.005:endPunkt
        [centers,~] = imfindcircles(image,[radius-floor(radius*0.19) ...
        radius+floor(radius*0.19)],'ObjectPolarity','dark', ...
        'Method','twostage','Sensitivity',i);
        foundCircles(arrayCounter) = length(centers);
        arrayCounter = arrayCounter + 1;
    end
    toc

```

```
    x = startPunkt+0.005:0.005:endPunkt;
    y = foundCircles(1:arrayCounter-1);
    differential = diff(y);

    indicesOfLocalMinima = islocalmin(differential);
    localMinima = x(indicesOfLocalMinima);

    % refine tf
    tf_refined = differential(indicesOfLocalMinima) <
                    ceil(mean(differential(indicesOfLocalMinima)));
    localMinima_refined = localMinima(tf_refined);

    sensitivityPoint = localMinima_refined(end);

    %% VISUALISIERUNG
    plot(x,differential,x(indicesOfLocalMinima),differential(indicesOfLocalMinima),'r*')
    title('Gradient of Found Circles')
    xlabel('Sensitivity')
    ylabel('New found circles with every step')
end
```

## A.3 CircumferenceCalculation.m

**Quelltext A.3:** Verfeinerung der lokalen Minima

```
classdef CircumferenceCalculation
    %ARC This class stores the data for valid arcs.
    %   The data is just the cut points of the arcs in degrees.
    %   And additionally the information if it is a start or end point

    properties
        centers
        radius
    end

    methods
        % Konstruktor
        function obj = CircumferenceCalculation(centers, radius)
            if nargin > 0 % checke, ob die Anzahl der uebergebenen Parameter groesser 0 ist
                if isnumeric(centers) && isnumeric(radius)
                    obj.centers = centers;
                    obj.radius = radius;
                else
                    error('Value must be numeric')
                end
            end
        end

        function circumferenceTable = calculateCircumferenceTable(obj)
            % Berechne die Entfernungen der Kreise zueinander
            distances = squareform(pdist(obj.centers));

            % Erzeuge Matrix zur Speicherung der Abstands-Winkel
            distanceAngles = ones(length(obj.centers),length(obj.centers));

            % Fuelle die Distanzwinkel-Matrix mit Werten
            for i = 1:size(distances,1)
```

```
33              for j = 1:size(distances,2)
34
35                  % Hole ersten Punkt
36                  p1 = obj.centers(i,:);
37
38                  % Hole zweiten Punkt
39                  p2 = obj.centers(j,:);
40
41                  % Berechne Abstandsvektor der beiden Punkte
42                  distanceVector = p2 - p1;
43
44                  % Erzeuge x-Einheitsvektor
45                  xe = [1 0];
46
47                  % Berechne Winkel zwischen Einheitsvektor und Distanzvektor
48                  angle = acosd(min(1,max(-1, xe(:).' * distanceVector(:) / norm(xe) ...
49                                  / norm(distanceVector) )));
50
51                  % Passe gegebenenfalls den Wert fuer den korrekten Quadranten an
52                  % 1: Die Phase befindet sich im dritten oder vierten Quadranten
53                  if distanceVector(2) > 0
54                      angle = 360 - angle;
55                  end
56                  % 2: Wenn der Kreis den Winkel zu sich selbst berechnet
57                  if distanceVector == 0
58                      angle = 0;
59                  end
60
61                  % Speichere den berechneten Wert in die Distanzwinkel-Matrix
62                  distanceAngles(i,j) = angle;
63              end
64          end
65
66          % Erzeuge ArcAnglesCalculation-Objekt
67          arcAnglesCalc = ArcAnglesCalculation(distances,distanceAngles,obj.radius);
68
69          % Berechne Bogenstuecke
70          arcAngleTable = arcAnglesCalc.calculateArcAngleTable();
71
72          % Erzeuge die Bogen-Tabelle zum Speichern der gemergeten Boegen
73          circumferenceTable(size(arcAngleTable,1),size(arcAngleTable,2)) = Arc;
74
75          % Extrahiere die Reihen der Tabelle in einen Objektarray
76          for i = 1:size(arcAngleTable,1)
77              circle = arcAngleTable(i,:);
78              % Berechne die zusammengefassten Bogenwinkel
79              mergedArcs = obj.mergeArcsForCircle(circle);
80              circumferenceTable(i,1:length(mergedArcs)) = mergedArcs;
81          end
82
83      end
84
85      function mergedArcs = mergeArcsForCircle(~, circle)
86          % Entferne die Arc(0,360) Eintraege
87
88          % Sortiere die Bogenstuecke nach Startpunkt und entferne 0/360er
89          j = 1;
90          arcsOfCircle1 = Arc.empty(length(circle),0);
91          for ii = 1:length(circle)
```

```
92              %if circle(ii).startPoint == 0 && circle(ii).endPoint == 360
93              %else
94                  arcsOfCircle1(j) = circle(ii);
95                  j = j + 1;
96              %end
97          end
98
99          [~, ind] = sort([arcsOfCircle1.startPoint]);
100         arcsOfCircle1_sorted = arcsOfCircle1(ind);
101         % An dieser Stelle koennte man die doppelten Eintraege entfernen
102
103         % MERGE ALGORITHMUS
104
105         % Speichere alle Startwerte in Array s (bereits sortiert)
106         s = zeros(1, length(arcsOfCircle1_sorted));
107         for i = 1:length(arcsOfCircle1_sorted)
108             s(i) = arcsOfCircle1_sorted(i).startPoint;
109         end
110         s(diff(s) == 0) = [];
111
112         % Speichere alle Startwerte in Array e und sortiere sie
113         e = zeros(1, length(arcsOfCircle1_sorted));
114         for i = 1:length(arcsOfCircle1_sorted)
115             e(i) = arcsOfCircle1_sorted(i).endPoint;
116         end
117         e = sort(e);
118         e(diff(e) == 0) = [];
119
120         % Entferne diejenigen Start- und Entpunkte, die im ueberlappten
121         % Bereich liegen
122         for i = 1:length(arcsOfCircle1_sorted)
123         if arcsOfCircle1_sorted(i).endPoint - arcsOfCircle1_sorted(i).startPoint > 0
124         s_index = s<arcsOfCircle1_sorted(i).startPoint | s>arcsOfCircle1_sorted(i).endPoint;
125         e_index = e<arcsOfCircle1_sorted(i).startPoint | e>arcsOfCircle1_sorted(i).endPoint;
126         else
127         s_index = s<arcsOfCircle1_sorted(i).startPoint & s>arcsOfCircle1_sorted(i).endPoint;
128         e_index = e<arcsOfCircle1_sorted(i).startPoint & e>arcsOfCircle1_sorted(i).endPoint;
129         end
130         s(s_index) = [];
131         e(e_index) = [];
132         end
133
134         % Fasse die Bogenstuecke zu zusammenhaengenden Stuecken zusammen
135         mergedArcs = Arc.empty(length(arcsOfCircle1_sorted),0);
136         k = 1;
137
138         for j = 1:length(e)
139             tempS = -1;
140             for i = 1:length(s)
141                 if s(i) > e(j)
142                     % StartPoint ohne Endpoint? Nehme naechsten Endpoint
143                     % nach dem Phasenuebergang mit
144                     if i == length(s) && tempS == -1
145                         mergedArcs(k) = Arc(s(i),e(1));
146                         k = k + 1;
147                     end
148                     continue
149                 else
150                     tempS = s(i);
```

```
151                     end
152                 end
153                 if tempS > -1
154                     mergedArcs(k) = Arc(tempS,e(j));
155                     k = k + 1;
156                 end
157             end
158 
159         end
160 
161     end
162 end
```

## B Rasterelektronenmikroskop Aufnahmen

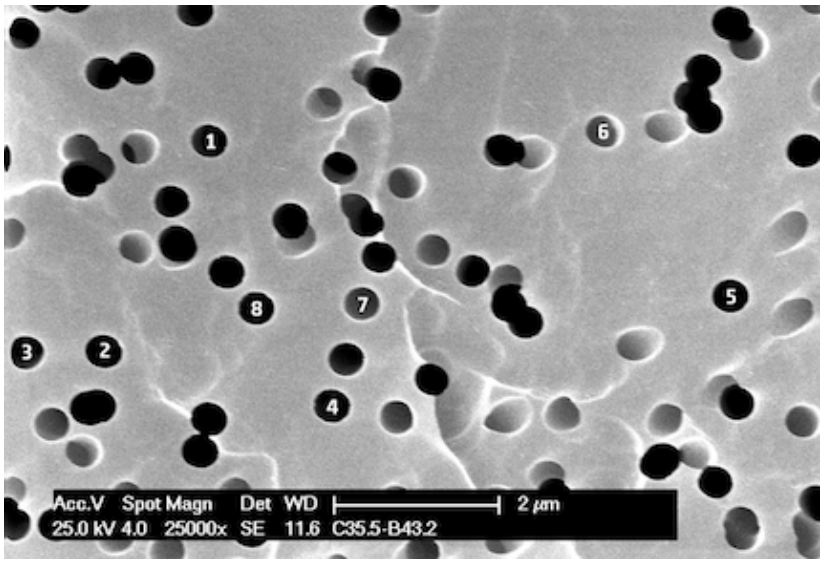



**Abbildung B.1.:** Mittlerer Kontrast und Helligkeit in 25.000-facher Vergrößerung

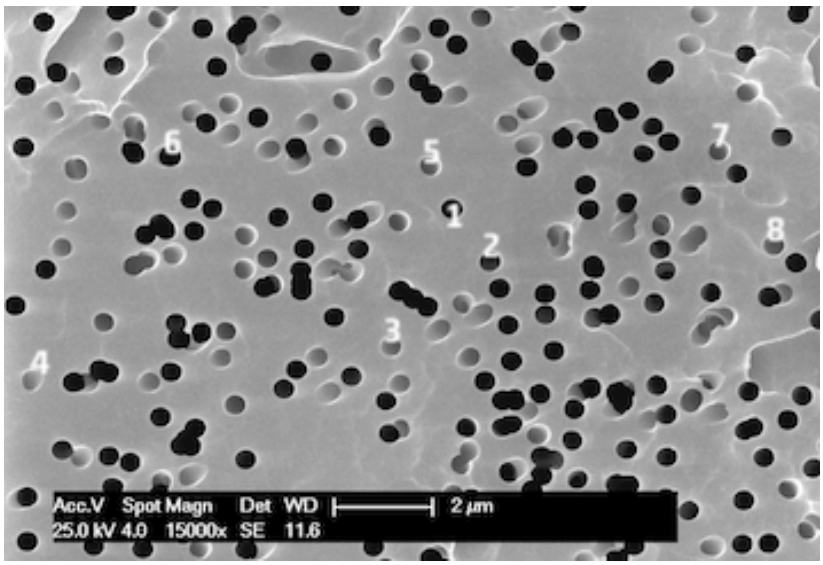



**Abbildung B.2.:** Mittlerer Kontrast und Helligkeit in 15.000-facher Vergrößerung

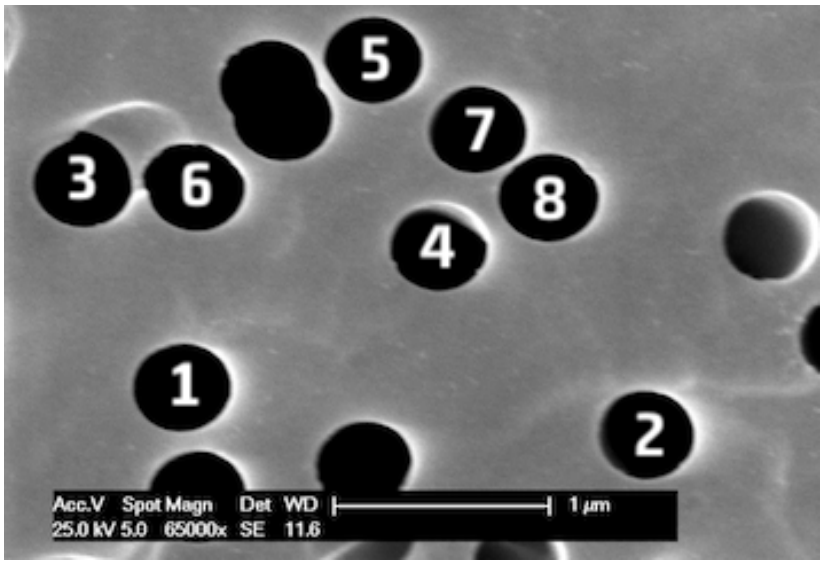



**Abbildung B.3.:** Mittlerer Kontrast und Helligkeit in 65.000-facher Vergrößerung