Lehrstuhl für Nachrichtentechnik

Univ.-Prof. Dr. H. Marko

Technische Universität München

Diplomarbeit

Thema:

Entwicklung eines rechnergestützten Verfahrens zur Mustererkennung in einem biologischen Präparat mit Mitteln der digitalen Bildverarbeitung

Betreuer:

Herr Dipl.-Ing. H. Platzer

Angefertigt von:

Axel Findling
Matr.-Nr.: 665829
Waldhornstraße 32c
8034 Germering

Bearbeitungsdauer: 6 Monate
Abgabetermin: 14. August 1991

Mit freundlicher Unterstützung des

Physiologischen Instituts der
Ludwig-Maximilians-Universität München
Im Besonderen:
Herrn Univ.-Prof. Dr. M. Wahl und Herrn Dr. Dr. L. Schilling

1.Inhaltsverzeichnis

2.Einleitung

2.1.Versuchsbeschreibung

2.1.1.Vorbereitung des biologischen Präparats

Die Versuche werden an lebenden Katzen durchgeführt. Die Tiere werden zunächst narkotisiert, so daß sie im weiteren Versuchsablauf keine Schmerzen empfinden können. Aus dieser Narkose werden sie nicht mehr erwachen, d. h. die Tiere müssen auch nach dem Versuch nicht unter Folgeschmerzen leiden.

Bei Versuchsbeginn wird die Luftröhre kanüliert, und in die Leistengefäße werden zwei arterielle und zwei venöse Katheter gelegt. Dabei dienen die arteriellen Katheter der Blutdruckmessung und den versuchsbegleitenden Blutabnahmen, die venösen Katheter dagegen zur Relaxation der Katze und zur Tracerinjektion in den Blutkreislauf. Relaxation bedeutet, daß keine Muskeltätigkeit mehr möglich ist. Die Tiere müssen daher maschinell beatmet werden. Als Tracer wird Fluoreszein benutzt, das je nach Versuch an verschieden große Dextranmoleküle gekoppelt ist. Der Tracer dient als Informationsträger für die Durchlässigkeit der Blutgefäße des Gehirns. Er wird durch Licht der Wellenlänge <490nm angeregt, und das emittierte Licht (Fluoreszenz) oberhalb von 525nm wird aufgezeichnet.

Ein in etwa quadratisches Stück des Schädelknochens der linken Schädelregion wird ausgebohrt und die darunter liegende harte Hirnhaut, die Dura, entfernt. Um die Hirnoberfläche vor dem Austrocknen zu schützen, wird sie mit Paraffinöl überschichtet.

2.1.2.Experimenteller Versuchsablauf

Auf die Gehirnoberfläche werden je nach Versuch verschiedene Teststoffe in aufsteigender Konzentration gegeben (superfundiert), die in künstlichem Liquor (Gehirnflüssigkeit) gelöst sind. Diese Stoffe sind z.B. Nikotin, Adenosin oder Histamin. Diese Substanzen diffundieren an die Gefäßwände der Hirngefäße heran und es lassen sich zwei unterschiedliche Reaktionen der Gefäße, abhängig jeweils von der verwendeten Substanz, beobachten: Dilatationen und Extravasationen.

Dilatation bedeutet eine Zunahme der Gefäßdurchmesser. Dabei bewirkt die verwendete Substanz eine Erschlaffung der Gefäßwandmuskeln, so daß sich der Gefäßdurchmesser vergrößert. Aufgrund des kleineren Strömungswiderstandes resultiert daraus eine höhere Durchblutung. Dieser Effekt wird hauptsächlich an Arterien beobachtet, da deren Gefäßwände mehr Muskelgewebe enthalten.

Als Extravasation bezeichnet man den Austritt des Tracers aus Gefäßen. Dieses ist ein Indikator für die Öffnung der Bluthirnschranke. Dabei öffnen sich, je nach Substanz, verschiedene funktionelle Poren und lassen daher verschieden große Tracer austreten, deren Größe abhängig von dem gewählten Dextranmolekül ist. Diese Reaktion läßt sich hauptsächlich an Venen beobachten, die eine dünnere Gefäßwand im Vergleich zu Arterien besitzen.

2.1.3.Technischer Versuchsablauf und Bildgewinnung

Das Beobachtungsareal auf der Gehirnoberfläche wird in vier bis sechs Felder unterteilt, die über einen programmierbaren in x und y-Richtung verfahrbaren Kreuztisch, auf dem das Untersuchungsobjekt liegt, angefahren werden können. Diese Felder, die durch eine Quecksilberlampe angeregt werden, werden jeweils beim inspiratorischen Plateau (d.i. ein von der Beatmungsmaschine erzeugte Konstanz des Beatmungsdruckes, die ca. 1.5sec lang ist) mit einer SIT-Kamera, die auf ein Zoom-Makroskop aufgesetzt ist, erfaßt und von einem U-Matic SP- Videorekorder aufgezeichnet.

Die zu Beginn der eigentlichen Meßphase ausgewählten Untersuchungsareale werden im weiteren Verlauf des Versuchs zu festgelegten Zeiten untersucht. Vor jedem Aufnahmezeitpunkt wird eine sogenannte 'Topfaufnahme' gemacht. Dabei wird ein mit Tracer gefülltes Gefäß aufgenommen, um die ungleichmäßige, annähernd kegelförmige, Lichtverteilung der Quecksilberlampe in der Versuchsauswertung berücksichtigen zu können. Zu jedem interessierenden Zeitpunkt werden also das Gefäß und die Felder aufgenommen, wobei jede Aufnahme in 16- und in 32-facher Vergößerung aufgezeichnet wird. Die stärker vergrößerten Aufnahmen (kleinerer Bildausschnitt) dienen dabei lediglich zur Messung der Gefäßdurchmesser.

Der erste Aufnahmezeitpunkt dient zum Aufzeichnen der Referenzbilder. Die Hirnoberfläche ist dabei nur von dem Paraffinöl bedeckt. Danach wird zunächst künstlicher Liquor superfundiert, um die Auswirkungen des Lösungsmittels alleine zu bestimmen. Die ausgewählten Areale werden nach einer, nach 15 und nach 30 Minuten aufgezeichnet. Nach dieser Zeit wird eine der oben genannten Substanzen, die in diesem Liquor gelöst ist, auf die Hirnoberfläche gegeben. Jede Konzentrationsstufe wird ebenfalls nach ein, 15 und nach 30 Minuten aufgezeichnet, bevor eine jeweils 10-fach höhere Konzentration superfundiert wird.

2.2.Aufgabenstellung

Ziel dieser Diplomarbeit ist es, mit mustererkennenden Verfahren Extravasationen zu erkennen und von Artefakten zu unterscheiden.

3.Systembeschreibung

3.1.Hardware

Die Bilder liegen analog aufgezeichnet als Videosequenzen auf einem U-Matic SP-Videoband vor.
Zur Weiterverarbeitung steht ein IBM-kompatibler AT-Personalcomputer vom Typ Hewlett-Packard Vectra RS/16 mit einem Intel 80368 Prozessor und einem 80387 Co-Prozessor zur Verfügung. Dieser Rechner arbeitet mit einer Taktfrequenz von 16 MHz und ist mit 120 MByte Festplattenkapazität und einem Bandlaufwerk mit 60 MByte Kapazität ausgestattet. Weiterhin besitzt der Rechner eine Bildkarte der Firma Itex vom Typ FG-100 und einen daran extern angeschlossenen RGB-Monitor der Firma Barco. Die Bildkarte hat einen Bildspeicher von 1024*1024 Bildpunkten mit einer Tiefe von 8 Bit. Weitere 4 Bit dienen als sogenannte Overlay-Bits, die zusätzliche Bildinformationen, wie z. B. die verwendete Look-Up-Table beinhalten können.

3.2.Software

Die Realisierung der Algorithmen wurde in der Programmiersprache C durchgeführt. Dabei wurde der Turbo C++ Compiler von Borland verwendet, der die Programme im Huge-Modell kompiliert.

3.3.Bildformat

Die Bilder liegen in einem Format von 503 Spalten * 512 Zeilen vor. Dapei kann jeder Pixel die Grauwerte von 0 bis 255 annehmen. Hierbei repräsentiert der Grauwert (die Intensität) 0 'dunkel' und 255 'hell'. Die Bildpunkte stehen in einem Seitenverhältnis von 1/1.543 (Höhe zu Breite). Die reale Pixelbreite beträgt bei einer 16-fachen Vergrößerung, in der die zu bearbeitenden Bilder vorliegen, 12.2 µm. Die entsprechende Pixelhöhe beträgt real 7.91 µm.
Ein Bild wird bei den meisten Berechnungen im RAM des Computers gehalten, um die Rechenzeit zu verkürzen. Bis zu vier weitere benötigte Bilder können dann im Bildspeicher der Bildkarte abgelegt werden.

4.Bildvorverarbeitung

4.1.Digitalisierung

Jeder Aufnahmezeitpunkt liegt als Videosequenz von ca. zwei Sekunden vor. Innerhalb dieser zwei Sekunden wird der stabilste Moment abgewartet, an dem dann vier aufeinanderfolgende Bilder durch die Bildkarte digitalisiert und in den Bildspeicher geschrieben werden. Diese vier Bilder werden nach Formel 4.1 linear gemittelt und als Datei auf der Festplatte gespeichert. Die Mittelung ist nötig, damit die Bewegung, z.B. der Blutkörperchen innerhalb der Gefäße, als möglichst homogene Fläche erscheint.

$$x^r_{i,j} = (x^1_{i,j}+x^2_{i,j}+x^3_{i,j}+x^4_{i,j}) / 4 \quad \text{für alle Bildpunkte } i,j ;$$

x^r : resultierendes Bild; $x^1 \ldots x^4$: digitalisierte Bilder

Formel 4.1: Bildmittelung

Im Folgenden werden nach jedem Kapitel Beispiele von bearbeiteten Bildern und zur Bearbeitung benötigte Bilder beigefügt sein, um die Bearbeitungssequenz verfolgen zu können. Als Beispielbilder wurden ein Bild ohne Extravasation und das selbe Bild zu einem späteren Zeitpunkt mit Extravasation gewählt. Bild 1 und Bild 2 zeigen diese digitalisierten und gemittelten Bilder.

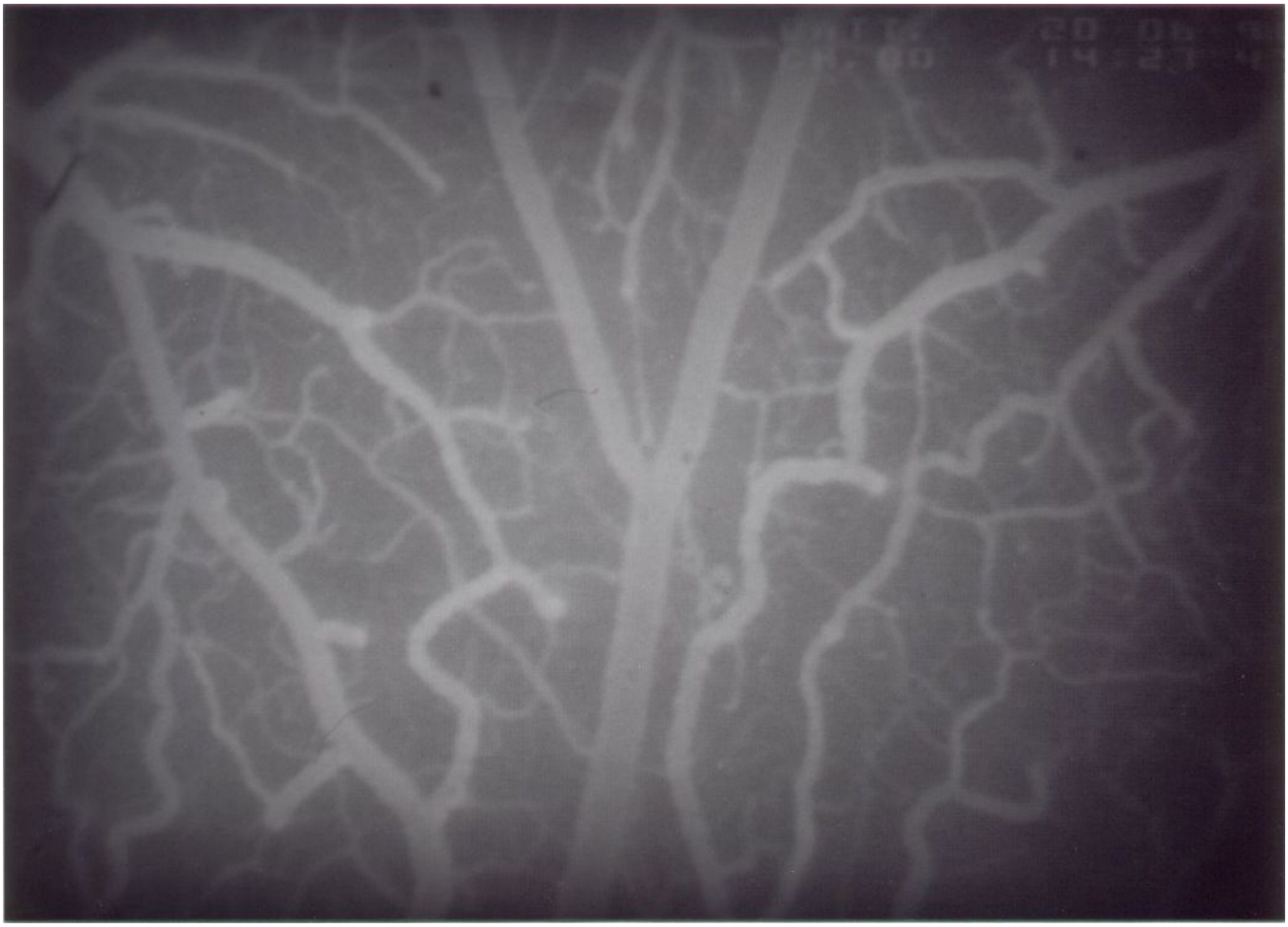


Bild 1: Digitalisiertes und gemitteltes Bild ohne Extravasation

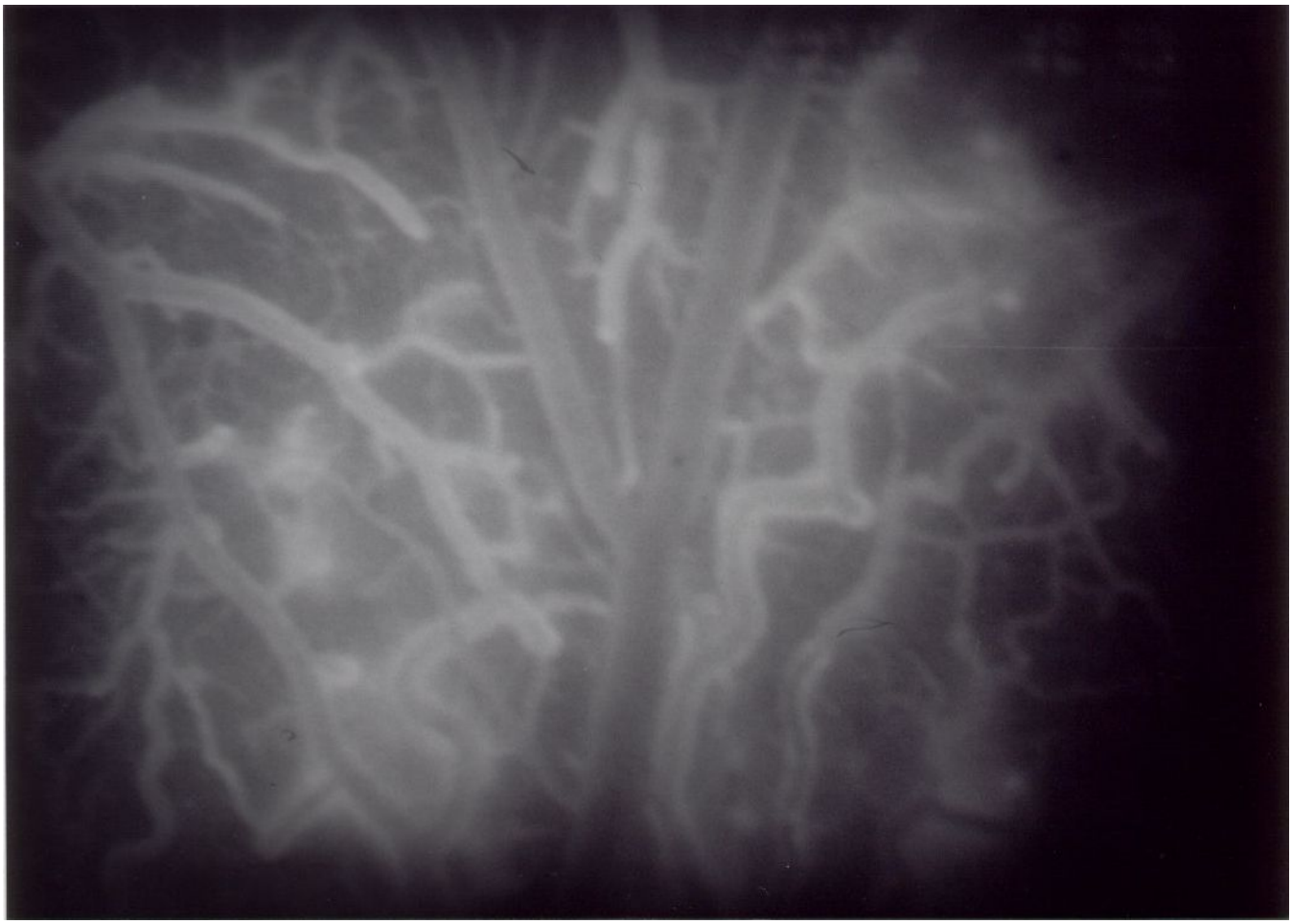


Bild 2: Digitalisiertes und gemitteltes Bild mit Extravasation

4.2.Shadingkorrektur

Die Quecksilberlampe zeigt bei einem neutralen Beobachtungsobjekt eine inhomogene Lichtverteilung, die mit den oben erwähnten 'Topfbildern' erfaßt werden kann. Um das Rauschen der 'Topfbilder' zu unterdrücken, werden sie zuvor noch mediangefiltert. Die Lichtverteilung zeigt etwa in der Mitte des Bildes ein deutliches Helligkeitsmaximum und nimmt zum Rand hin ab. Diese Bilder werden als Korrekturbilder zu den aufgenommenen Bildsequenzen verwendet, um eine gleichmäßige Helligkeitsverteilung zu erhalten. Dazu wird zunächst der maximale Helligkeitswert x^{max} des betreffenden Korrekturbildes bestimmt. Die korrigierten Bilder berechnen sich dann nach folgender Formel 4.2:

$$x^r_{i,j} = (x_{i,j} * x^{max}) / x^k_{i,j} \quad \text{für alle Bildpunkte i,j ;}$$

x^r : korrigiertes Bild; x : zu korrigierendes Bild; x^k : Korrekturbild

Formel 4.2 : Shadingkorrektur

Siehe dazu auch Kapitel 7.1..

In Bild 3 ist ein mediangefiltertes Topfbild zu sehen, die Bilder 4 und 5 zeigen die Bilder 1 und 2 jeweils shadingkorrigiert.

Bild 3: Mediangefiltertes 'Topfbild'

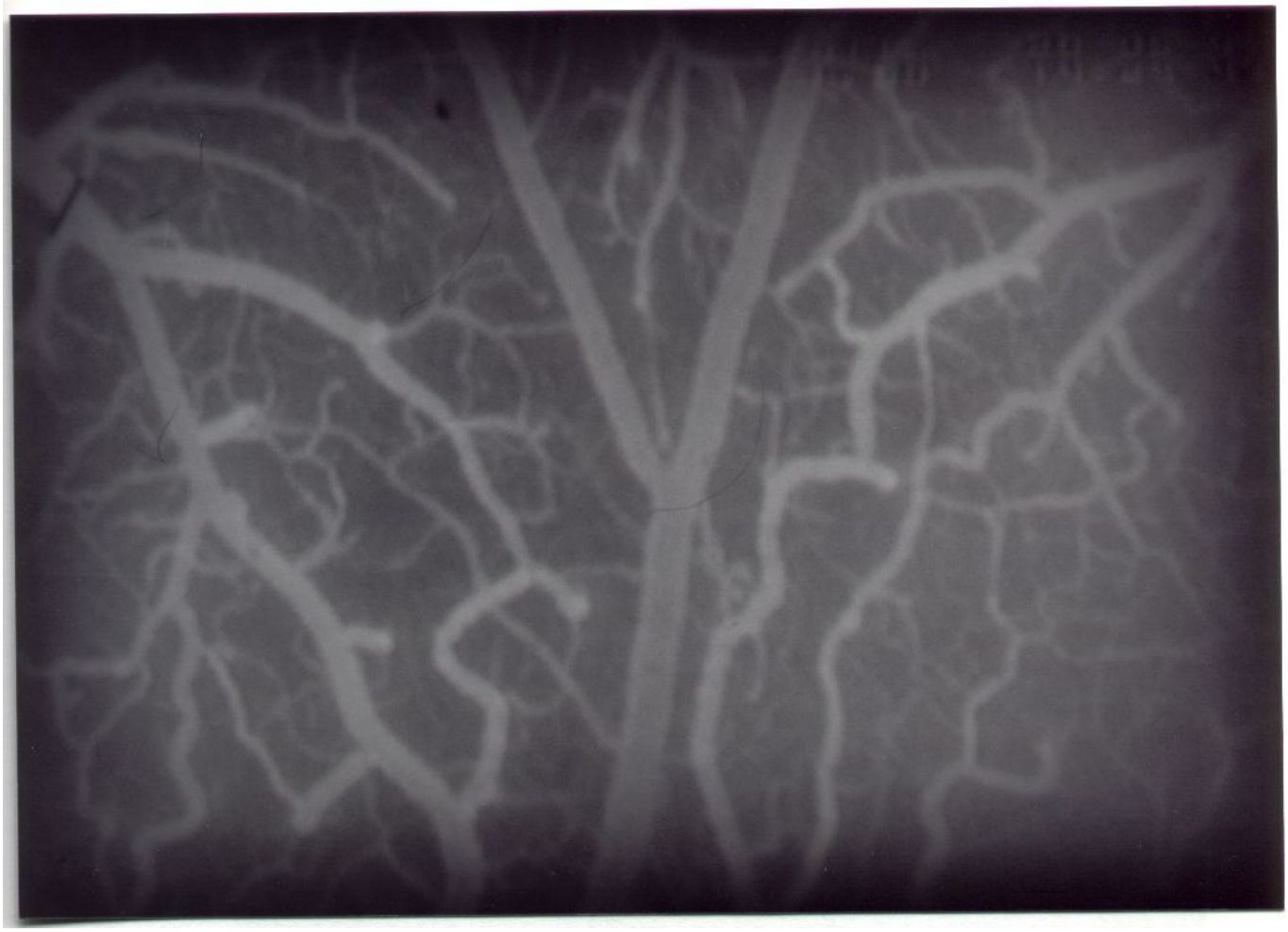


Bild 4: Bild 1 Shadingkorrigiert

Bild 5: Bild 2 Shadingkorrigiert

4.3.Lagekorrektur

Die soweit bearbeiteten Bilder müssen nun zueinander ausgerichtet werden, das heißt, um ein Subtraktionsbild erstellen zu können, müssen die Aufnahmen gleicher Felder auf das Bezugsbild (Ölaufnahme) ausgerichtet werden. Dieses geschieht mit Hilfe des von Lenz [1] entwickelten Lagekorrekturprogrammes mit Namen 'estpar', das für diesen Anwendungsfall umgeschrieben im Anhang beigefügt ist.
Das Programm muß iterativ aufgerufen werden, wobei sich als zeitlich vertretbar und genügend genau folgende Kriterien bewährt haben: Eine Rechnung mit Ordnung zwei ist ausreichend. Die Berechnungen sollen abbrechen, wenn die lineare Verschiebung in x- und in y-Richtung betragsmäßig weniger als je 0.15 Pixel betragen, oder maximal 20 Iterationen gerechnet wurden.
Jedes Bild wird auf das zeitlich direkt vorangegangene Bild korrigiert, um zum einen die zu berechnenden Verschiebungen so gering wie möglich zu halten, und damit sich zum anderen die Bildinhalte der zu vergleichenden Bilder am wenigsten unterscheiden. Die endgültigen Verschiebungsparameter ergeben sich aus den neu berechneten. Diese werden mit denen des Vorgängerbildes verknüpft. So erhält man die Verschiebungsparameter, die das neue Bild mit dem Bezugsbild zur Deckung bringen. Die durch die Verschiebung neu hinzukommenden Flächen werden zu Null gesetzt.

Siehe dazu auch Kapitel 7.2..

4.4.Subtraktionsbild erstellen

Da im Regelfall die Helligkeit der Bildfolgen zunehmen, wird das Bezugsbild von den folgenden lagekorrigierten Bildern abgezogen, damit bei der Subtraktion möglichst wenig Information verlorengeht. Negative Werte werden bei der Subtraktion zu Null gesetzt. Dies darf man machen, da mögliche Extravasationen einen Helligkeitsanstieg bedeuten, der nicht im Bezugsbild vorkommen kann. Somit berechnet sich das Subtraktionsbild wie folgt:

$$x^r_{i,j} = x_{i,j} - x^b_{i,j} \quad \text{für alle Bildpunkte i,j mit } (x_{i,j} - x^b_{i,j}) >= 0 ;$$

$$x^r_{i,j} = 0 \quad \text{für alle Bildpunkte i,j mit } (x_{i,j} - x^b_{i,j}) < 0 ;$$

x^r : Subtraktionsbild ; x^b : Bezugsbild ; x : zu analysierendes Bild

Formel 4.3: Subtraktionsbildberechnung

Siehe dazu auch Kapitel 7.3..

Bild 6 zeigt hier das shadingkorrigierte Referenzbild, das von den Bildern 5 und 6 nach der Lagekorrektur abgezogen wird. Die Bilder 7 und 8 zeigen das Ergebnis.

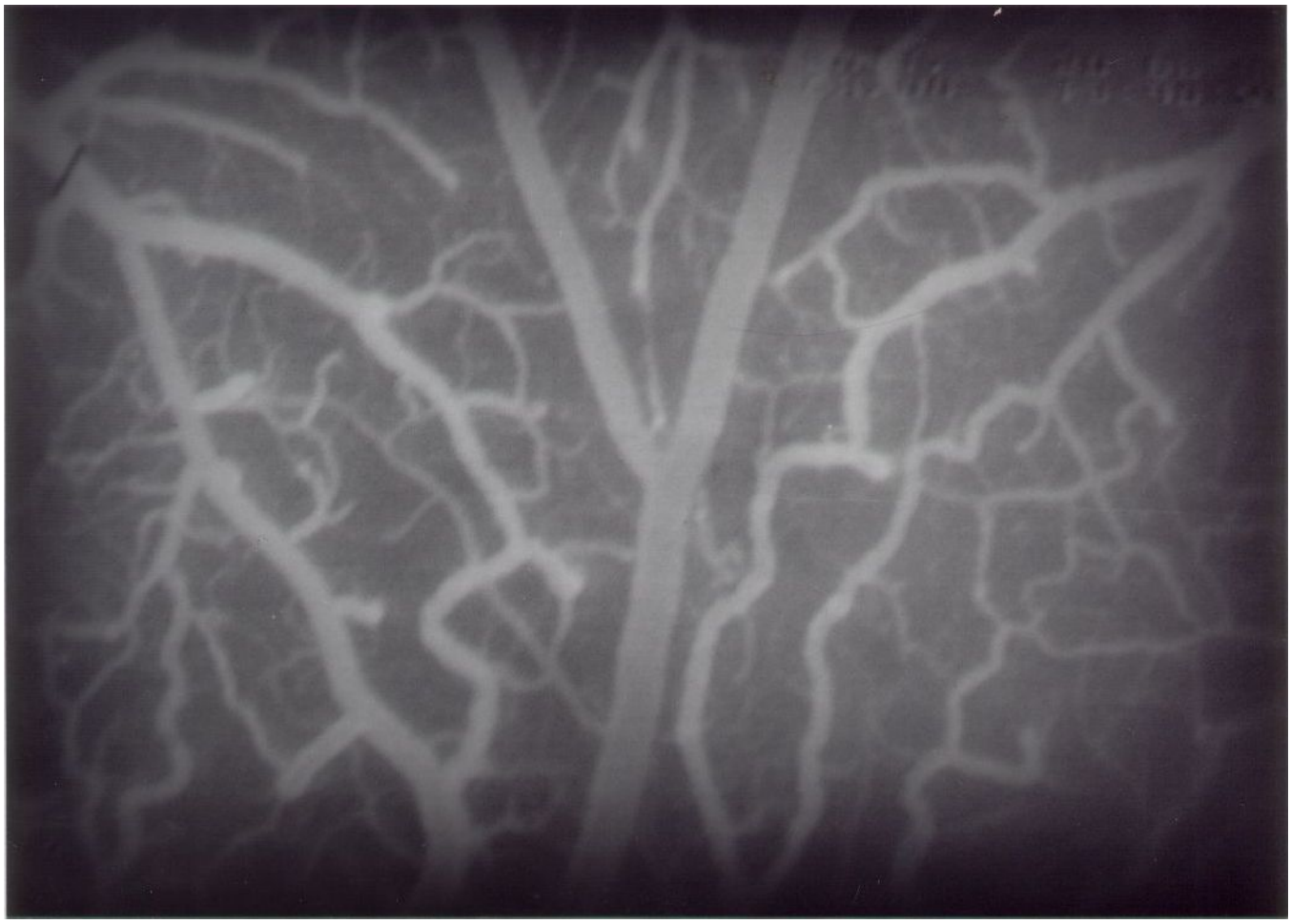


Bild 6: Shadingkorrigiertes Referenzbild

Bild 7: Subtraktionsbild zu Bild 4

Bild 8: Subtraktionsbild zu Bild 5

4.5.Rauschunterdrückung

Um Störungen durch Rauschen möglichst gering zu halten werden zwei Verfahren angewendet, um zum einen das statistische Rauschen und zum anderen das Grundrauschen bei völlig abgedunkeltem Objekt zu vermindern.

4.5.1.Medianfilterung

Die Medianfilterung dient dazu, das statistische Rauschen zu unterdrücken. Dabei wird aus den acht Nachbarpunkten und dem betreffenden Punkt selber derjenige Grauwert als neuer Pixelgrauwert ermittelt, der genau in der Mitte der neun sortierten Grauwerte liegt. Dies wird auch als Medianfilterung mit einem 3*3-Kernel bezeichnet. Möglicherweise auftretende Helligkeitskanten und -rampen werden dabei nicht verwischt.

Siehe dazu auch Kapitel 7.4..

4.5.2.Unterdrückung des Grundrauschens

Aus vorangegangenen Untersuchungen wurde ermittelt, daß das Grundrauschen im Grauwertebereich von Null bis fünf liegt. Aus diesem Grund werden alle Grauwerte dieses Bereichs im Subtraktionsbild zu Null gesetzt.

Siehe dazu auch Kapitel 7.5..

Die folgenden Bilder 9 und 10 zeigen die mit den obigen rauschmindernden Verfahren bearbeiteten Bilder 7 bzw. 8.

Bild 9: Bearbeitetes Subtraktionsbild Bild 7

Bild 10: Bearbeitetes Subtraktionsbild Bild 8

5.Regionenerfassung

Ausgehend von dem Subtraktionsbild wird dieses nun in Regionen eingeteilt, die voneinander unterscheidbar sind. Diese Regionen werden nach verschiedenen Kriterien vermessen, und es erfolgt aufgrund noch zu ermittelnder Bewertungskriterien eine Zuordnung, ob das Objekt eine Extravasation darstellt oder nicht.

5.1.Regionenabgrenzung

Um eine eindeutige Regioneneinteilung durchführen zu können, müssen einzelne für sich stehende oder objektverbindende Pixel ungleich Null eliminiert werden. Dazu wird das Bild horizontal, vertikal und in den zwei Hauptdiagonalen nach zusammenhängenden Pixelstrecken durchsucht, die genau ein Pixel lang sind. Diese werden dann gelöscht. Man kann davon ausgehen, daß Extravasationen nicht durch solche Pixel repräsentiert werden, deshalb stellt diese Abgrenzung keinen Verlust an Information diesbezüglich dar.

Siehe dazu auch Kapitel 7.6..

5.2.Regioneneinteilung

Das Bild wird von der linken oberen Ecke zeilenweise bis zur rechten unteren Ecke nach Grauwerten ungleich Null dursucht. Hat das Programm einen solchen gefunden, beginnt es in einer Kette alle vier vertikalen und horizontalen Nachbarn zu untersuchen, und falls diese auch ungleich Null sind, in einer Kette zu speichern. Die Kette wird dann bis zum Ende durchlaufen. Dadurch, daß das Programm das Ergebnis in einem neuen Bild abspeichert, wird kein mehr als einmal bearbeitet. Die so entstandene zusammenhängende Region wird numeriert, indem alle Pixel dieser Region auf den Numerierungswert im neu angelegten Bild gesetzt werden. Danach wird die nächste Region gesucht. Da die Bilder nur acht Bit Tiefe besitzen, können maximal (ohne nullgesetzten Hintergrund) 255 Objekte pro Bild eindeutig unterschieden werden. Damit hier kein Überlauf auftritt, werden pro Bild der mittlere Grauwert aller Regionen ermittelt. Alle Grauwerte, die kleiner als dieser Mittelwert sind, werden zu Null gesetzt. Außerdem werden nur solche Regionen bearbeitet, die kleiner als 25 Pixel sind, da durch die interaktive Zuordnung von Regionen zu Extravasationen, auf die später noch eingegangen wird, kleinere Flächen nicht erkannt werden können.

Siehe dazu auch Kapitel 7.7..

Bild 11 zeigt das in Regionen unterteilte Bild 10, Bild 12 die Konturen dazu. Das in Regionen unterteilte Bild 9 ist hier nicht abgebildet, da es nur drei kleine Regionen enthält, der maximale Grauwert 3 aber nicht mit dem Auge erkennbar ist.

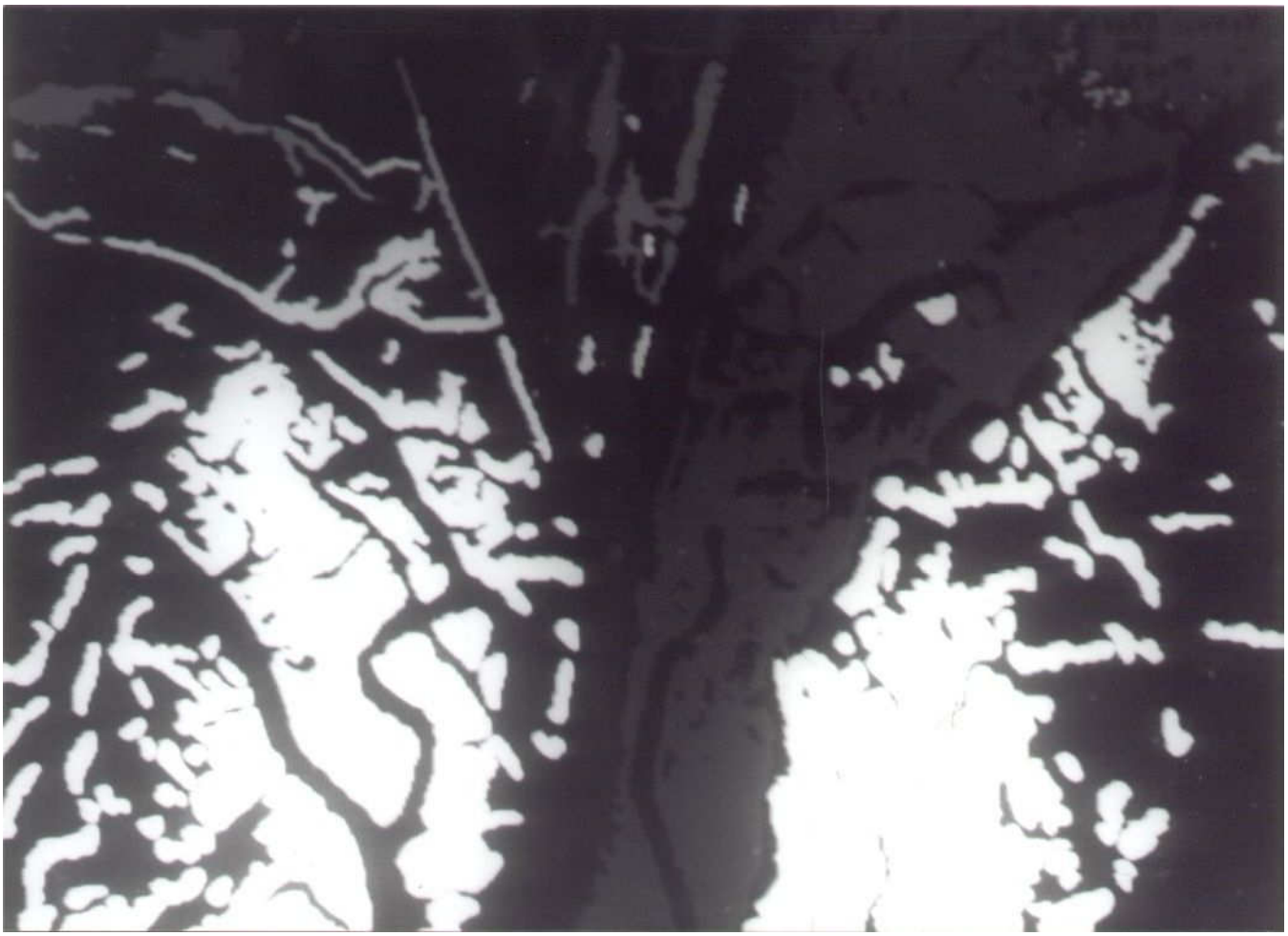


Bild 11: In Regionen unterteiltes Bild 10

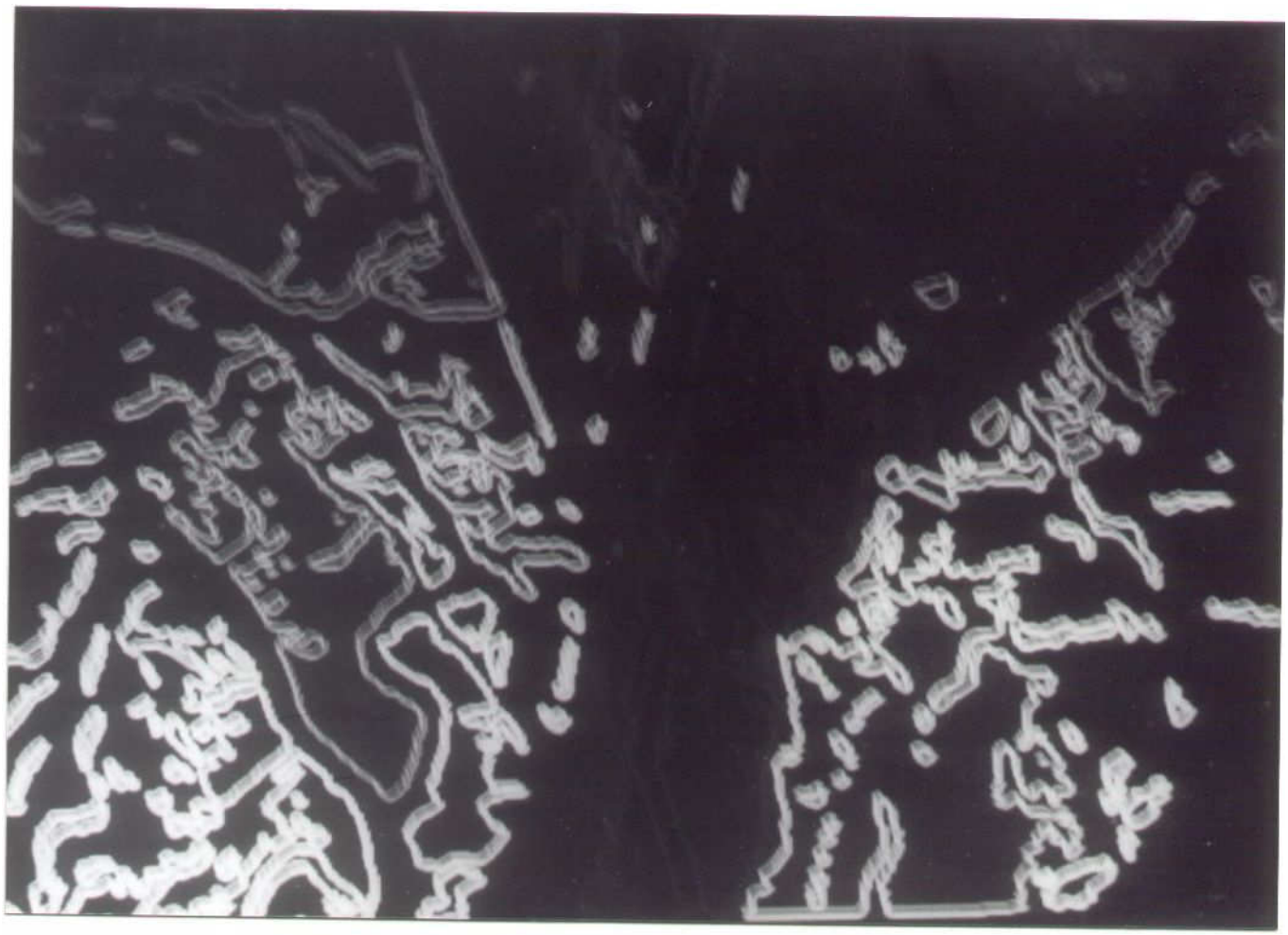


Bild 12: Konturen des Bildes 11

5.3.Regionenvermessung

5.3.1.Grauwertabhängige Größen

Gleichzeitig mit der Regioneneinteilung werden die grauwertabhängigen Charakteristika der Regionen gemessen, da hier das Subtraktionsbild noch mit den originalen Grauwertdaten vorliegt.

5.3.1.1.Mittlere Intensität

Um die mittlere Intensität zu bestimmen, wird der Mittelwert über alle Pixel der betreffenden Region nach folgender Formel 5.1 bestimmt:

I_r^{mittel} = (Summe aller Grauwerte der Region r) /
(Anzahl aller Grauwerte der Region r);
I_r^{mittel} : mittlere Intensität der Region r
Formel 5.1 : Berechnung der mittleren Intensität

Die mittlere Intensität kann Werte zwischen 0.0 und 255.0 annehmen.

Siehe dazu auch Kapitel 7.7..

5.3.1.2.Maximale Intensität

Die maximale Intensität ist derjenige Grauwert eines Pixels einer Region, der den größten Wert besitzt. Dieser Wert kann eine ganzzahlige Zahl zwischen 0 und 255 sein.

Siehe dazu auch Kapitel 7.7..

5.3.2.Formabhängige Größen

Die formabhängigen Größen berechnen sich allein aus der Form der Regionen. Zur Berechnung der einzelnen Größen werden für Fläche und Umfang die mit dem numerierten Wert gefüllten Regionen herangezogen. Für die anderen Größen wird ein Bild benutzt, bei dem lediglich die Konturen der Regionen erkennbar sind. Zur Konturen-Extration werden alle Pixel zu Null gesetzt, die ungleich Null sind und deren vertikale und horizontale Nachbarn Intensitäten ungleich Null besitzen.

5.3.2.1.Fläche

Zur Flächenberechnung werden die Pixel einer jeden Region gezählt und mit der realen Pixelhöhe und der realen Pixelbreite laut Formel 5.2 multipliziert. Die Einheit der durch das Programm berechneten Größe beträgt 100µm².

A_r = (Anzahl aller Pixel der Region r) * Pixelhöhe * Pixelbreite;

A_r : Fläche der Region r

Formel 5.2 : Flächenberechnung

Siehe dazu auch Kapitel 7.7..

5.3.2.2.Umfang

Zur Umfangsberechnung werden die Übergänge zweier benachbarter Pixel betrachtet. Bei einem Übergang von vertikal benachbarten Pixeln, von denen einer die Intensität Null und der andere zu einer Region gehört, wird die Pixelbreite zum Umfang addiert. Sind zwei solche Pixel horizontal benachbart, so wird stattdessen die Pixelhöhe zum Umfang addiert. Die Einheit des Umfangs beträgt 10µm.

Siehe dazu auch Kapitel 7.8..

5.3.2.3.Rundheit

Das Rundheitsmaß wird nach einer Formel nach Liedtke/Ender ([2]; Seite 75) berechnet. Dabei muß zunächst ein die Region umschreibendes Achteck gefunden werden. Hierbei werden zunächst die acht Punkte der Kontur ermittelt, die auf den die Region umschließenden vertikalen, horizontalen und 45°-diagonalen Geraden liegen. Aus diesen werden die Eckpunkte des Achtecks berechnet und daraus wiederum, unter Berücksichtigung der realen Streckenverhältnisse, die Längen der Achteckseiten. Daraus und mit Hilfe der noch bekannten Fläche kann nun ein Maß für die Rundheit nach folgender Formel 5.3 abgeleitet werden. Der Zahlenwert ist einheitslos.

$$M_r^{rund} = 4*Pi*A_r / (\text{Umfang des die Region r umschießenden Achtecks})^2;$$

M_r^{rund} : Rundheit der Region r ; A_r : Fläche der Region r ; Pi : Kreiszahl

Formel 5.3: Berechnung der Rundheit einer Region

Siehe dazu auch Kapitel 7.9..

5.3.2.4.Länglichkeit

Das Maß für die Länglichkeit wird ebenfalls nach einer Formel nach Liedtke/Ender ([2]; Seite 75) berechnet. Die Berechnung des umschreibenden Achtecks erfolgt wie bei der Bestimmung der Rundheit. Aus dem Achteck kann dann mit folgender Formel 5.4 das Maß für die Länglichkeit abgeleitet werden. Wie bei der Rundheit ist auch hier der Zahlenwert einheitslos.

$$M_r^{lang} = Max\{(d_1+d_5)/(d_3+d_7);(d_2+d_6)/(d_4+d_8);$$
$$(d_3+d_7)/(d_1+d_5);(d_4+d_8)/(d_2+d_6)\};$$

M_r^{lang} : Länglichkeit der Region r ;

d_i : Seiten des die Region r umschließenden Achtecks im Uhrzeigersinn numeriert

Formel 5.4: Berechnung der Länglichkeit

Siehe dazu auch Kapitel 7.10..

5.3.2.5.Verwinkeltheit

Die Verwinkeltheit gibt ein Maß an, wie 'zerfurcht' die Kontur einer Region ist, d.h. ob die Kontur einer Region starke Richtungsänderungen aufweist. Es werden dabei die Richtungsänderungen, ausgedrückt in Bogenmaß, betragsmäßig aufaddiert, so daß sich für eine einseitig gekrümmte Kontur, also z.B. nur rechtsgekrümmt, zwangsläufig ein Wert von 2*Pi ergibt. Die Winkelberechnung erfolgt abhängig von den realen Strecken-verhältnissen.

Siehe dazu auch Kapitel 7.11..

6.Klassifikation

6.1.Wahl von charakteristischen Testbildern

Testbilder wurden aus zwei Versuchen vom 20.06.91 und vom 28.06.91 am Physiologischen Institut ausgewählt. Diese Versuche waren vom Verlauf gut gelungen, so daß Extravasationen erst spät auftraten. Aus diesen Versuchen wurden jedoch nur die Bilder verwendet, bei denen entweder keine Extravasation, oder punktuell lokalisierbare Extravasationen auftraten. D.h. Bilder mit sehr stark ausgeprägten Extravasationen, die das gesamte Bild umfaßten, wurden verworfen. So wurden rund 100 Bilder als Testbilder herangezogen. Darin waren 5900 Regionen, die keine Extravasation darstellten, und 139 Regionen mit Extravasationen.

6.2.Spezifizierung von Extravasationsregionen

Zur Entscheidungsfindung wie sich Extravasationsregionen darstellen, müssen diese Regionen zunächst in den Testbildern interaktiv gekennzeichnet werden. Dazu wird zu dem Original- und dem Subtraktionsbild ein drittes Bild erzeugt, das das lagekorrigierte Originalbild enthält, dem die Konturen der mit obigem Algorithmus gefundenen Regionen überlagert wurden. Durch Betrachten der drei Bilder muß nun entschieden werden, welche der erkannten Regionen eine Extravasationsregion darstellt. Dies geschieht durch ein Programm, bei dem man die Bilder betrachten und den Mauszeiger über das Bild führen kann, um mit ihm erkannte Extravasationsregionen 'anzuklicken'. Die Koordinaten werden gespeichert, um bei der Regionenauswertung gleich die Zuordnung zu 'Extravasation' oder 'keine Extravasation' vornehmen zu können.

6.3.Statistische Auswertung

Zu den sieben oben genannten Beschreibungskriterien wird zusätzlich ein achtes Kriterium aus der Division von Umfang zur Fläche mit der Einheit $10^{5*1/m}$ ermittelt.

Die ermittelten Regionendaten werden nun mit Hilfe von Microsoft-Excel in zwei getrennten Tabellen gespeichert, von denen die eine (DATEN1.TXT) die Daten der Extravasationsregionen und die andere (DATEN2.TXT) die Daten aller anderen Regionen in Spalten geordnet enthält. Diese Tabellen bilden die Grundlage für folgende Statistikprogramme:

AUSWERT.C : Dieses Programm ist das komplexeste. Es berechnet Häufigkeitshistogramme, Häufigkeitsdichtefunktionen, kann die Daten logarithmieren, stellt die Funktionen graphisch am Bildschirm dar und gibt die Ergebnisse auch als Tabellenform aus.
Siehe dazu auch Kapitel 7.12..

ZUS_ODER.C : Verwendet die Ergebnisse, die mit AUSWERT.C gefunden wurden, und verbindet die Kriterien mit einer ODER-Verknüfung. Dabei werden Erkennungsgrad und Fehlerquote berechnet.
Siehe dazu auch Kapitel 7.13..

ZUS_UND.C : Leistet das selbe wie ZUS_ODER.C, berechnet aber eine UND-Verknüpfung der Kriterien.
Siehe dazu auch Kapitel 7.14..

ZUS_KET.C : Verknüpft zwei Kriterien kettenförmig, d.h. es berechnet die Wirkungsweise eines zweiten Kriteriums, wenn das erste bereits angewendet wurde.
Siehe dazu auch Kapitel 7.15..

6.3.1. Auswertung einzelner Kriterien

Mit Hilfe des Programmes AUSWERT.C (siehe Kapitel 7.12.) werden die Daten der einzelnen Beschreibungskriterien ausgewertet. Da nur positive Werte in den Datentabellen vorliegen, kann für die Auswertung eine logarithmische Darstellung der Daten gewählt werden. Dabei wird der Tabellenwert nach folgender Formel 6.1 umgerechnet:

$$\text{'Zur Auswertung herangezogener Wert'} = \log_{10}(\text{Tabellenwert});$$

Formel 6.1: Logarithmieren der Tabellendaten

Die Daten werden in einem Histogramm zu 70 Säulen aufgetragen, wobei die Höhe jeder Säule die Anzahl der in ihrem Intervall liegenden Daten repräsentiert. Jedes Diagramm enthält dabei das Histogramm der Extravasationsregionen und das der anderen Regionen. Da die absolute Häufigkeit dieser zwei Datengruppen so unterschiedlich sind, wurden die Histogramme in der Höhe angeglichen. Histogramm Nr.1 (rot) repräsentiert die Extravasationsregionen und Histogramm Nr.2 (grün) die anderen Regionen.

In den entsprechenden Farben sind in einem weiteren Diagramm die Häufikeitsdichtefunktionen gezeichnet. Hier wurden die 0%, die 50% und die 100% als waagrechte Geraden eingezeichnet.

Die Tabellenauswertung enthält sieben Spalten, von denen die siebte Spalte den Kriterienwert (bzw. den logarithmierten Kriterienwert) enthält, zu dem die Häufigkeiten in den sechs vorhergehenden Spalten berechnet werden.
Die erste Spalte ('1'<=) beinhaltet die relative Häufigkeit aller Extravasationsregionen, die kleiner oder gleich dem Kriterienwert sind, bezogen auf die Gesamtzahl der Extravasationsregionen.
Die zweite Spalte ('1'>) berechnet sich zu 100% minus die erste Spalte, d.h die relative Häufigkeit aller Extravasationsregionen, die größer als der Kriterienwert sind, bezogen auf die Gesamtzahl der Extravasationsregionen.
In der dritten ('2'<=) und in der vierten Spalte ('2'>) stehen die entsprechenden relativen Häufigkeiten der anderen Regionen bezogen auf die Gesamtzahl dieser Regionen.

Um der stark unterschiedlichen absoluten Häufigkeit der zwei Datengruppen Rechnung zu tragen, wurden noch folgende zwei Häufigkeiten eingeführt:
Die fünfte Spalte ('p1'<=) zeigt die relative Häufigkeit der Extravasationsregionen, die kleiner oder gleich dem Kriterienwert sind, bezogen auf alle Regionen, die kleiner oder gleich dem Kriterienwert sind.
Die sechste Spalte ('p1'>) schließlich enthält die relative Häufigkeit der Extravasationsregionen, die größer als der Kriterienwert sind, bezogen auf alle Regionen, die größer sind als der Kriterienwert.
Falls in den zuletzt erwähnten Spalten der Wert 999.00% auftritt, bedeutet dies, daß hier eine Division durch Null stattfindet, d.h. daß keine Region existiert, die die Bedingung erfüllt.

In der ersten Tabelle sind in 20 Intervallschritten über den möglichen Bereich die Häufigkeiten eingetragen.

Die Bedingung zur Erkennung von Extravasationen muß lauten: Bei einer beliebig herausgegriffenen Region mit bestimmbaren Eigenschaften muß die absolute Häufigkeit von Extravasationsregionen mit diesen Eigenschaften mindestens so groß sein, wie die absolute Häufigkeit von anderen Regionen. D.h. die fünfte oder sechste Spalte müssen in einem bestimmten Bereich die 50%-Marke erreichen und überschreiten. Falls es solch einen Bereich gibt, sind die Häufigkeiten am 50%-Übergang in der zweiten Tabelle näher untersucht. Der Grenzwert wurde darin unterstrichen.

Zur Erkennung anderer Regionen ist eine entsprechende Regel ungeeignet, da die absolute Häufigkeit dieser Regionen die der Extravasationsregionen bei weitem übertrifft. Hier bestimmt man zu einer maximalen Grenze von 2-3% der relativen Häufigkeit der Extravasationsregionen (Spalten 1 und 2) die größtmögliche maximale relative Häufigkeit der anderen Regionen (Spalten 3 und 4). Der Bereich in dem dies zutrifft ist in der letzten Tabelle dargestellt. Der Grenzwert ist auch hier unterstrichen.

6.3.1.1.Maximale Intensität

Regionen mit einer maximalen Intensität größer als $10^{1.925} = 84.14$ sind zu 50% Extravasationen. Dies ist der Fall für 2.88% aller Extravasationen, d.h. dieses Kriterium ist nicht sehr geeignet für eine Entscheidung.

18.19% der Nicht-Extravasationen besitzen eine maximale Intensität kleiner oder gleich $10^{1.220618} = 16.62$. Nur 2.16% aller Extravasationen fallen in diesen Bereich.

Die folgenden vier Seiten enthalten die Auswertung dieser Größe.

Datenbezeichnung: Maximum; Anzahl der Daten 1: 139
Datenbezeichnung: Maximum; Anzahl der Daten 2: 5900
70 Säulen Logarithmische Darstellung
Reales Minimum: 0.903090 Reales Maximum: 2.361728
Minimum im Diagramm: 0.903090 Maximum im Diagramm: 2.382566
Intervallbreite: 0.020838
Histogramm 1 (rot) : Maximum = 12
Histogramm 2 (gruen) : Maximum = 457
Angeglichene Histogramme

Anfangswert: 0.903090 Endwert: 2.361728
Intervallbreite: 0.072932

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
0.00%	100.00%	0.15%	99.85%	0.00%	2.31%	0.903090
0.00%	100.00%	0.63%	99.37%	0.00%	2.32%	0.976022
0.00%	100.00%	3.10%	96.90%	0.00%	2.37%	1.048954
0.00%	100.00%	8.07%	91.93%	0.00%	2.50%	1.121886
1.44%	98.56%	14.64%	85.36%	0.23%	2.65%	1.194818
5.04%	94.96%	26.46%	73.54%	0.45%	2.95%	1.267749
8.63%	91.37%	39.61%	60.39%	0.51%	3.44%	1.340681
17.27%	82.73%	55.53%	44.47%	0.73%	4.20%	1.413613
32.37%	67.63%	71.66%	28.34%	1.05%	5.32%	1.486545
44.60%	55.40%	83.83%	16.17%	1.24%	7.47%	1.559477
49.64%	50.36%	91.75%	8.25%	1.26%	12.57%	1.632409
63.31%	36.69%	96.75%	3.25%	1.52%	20.99%	1.705341
79.86%	20.14%	99.03%	0.97%	1.86%	32.94%	1.778273
89.21%	10.79%	99.63%	0.37%	2.07%	40.54%	1.851205
96.40%	3.60%	99.90%	0.10%	2.22%	45.45%	1.924136
97.84%	2.16%	99.97%	0.03%	2.25%	60.00%	1.997068
98.56%	1.44%	100.00%	0.00%	2.27%	100.00%	2.070000
98.56%	1.44%	100.00%	0.00%	2.27%	100.00%	2.142932
98.56%	1.44%	100.00%	0.00%	2.27%	100.00%	2.215864
98.56%	1.44%	100.00%	0.00%	2.27%	100.00%	2.288796
99.28%	0.72%	100.00%	0.00%	2.29%	100.00%	2.361728

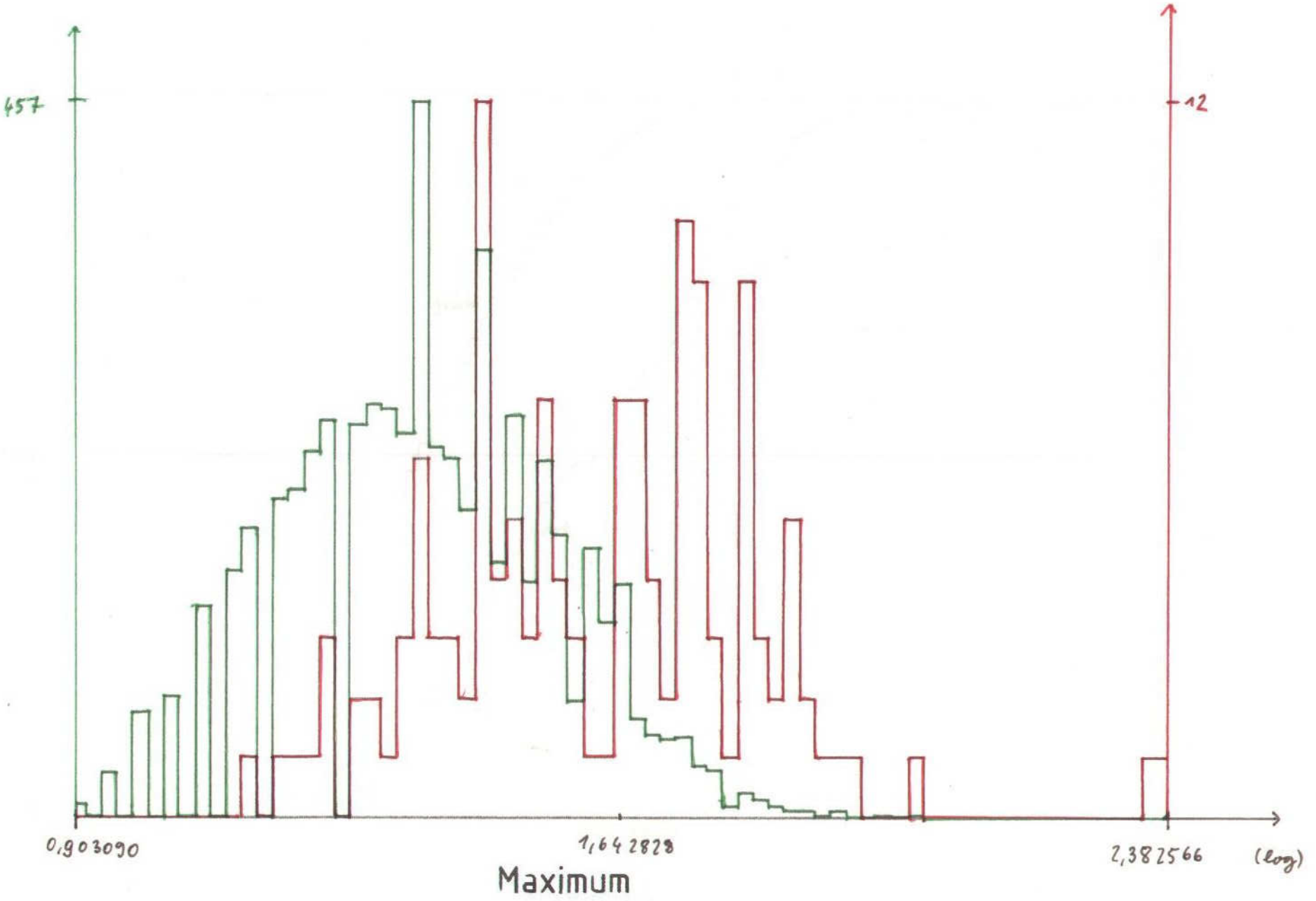
Anfangswert: 1.900000 Endwert: 2.000000
Intervallbreite: 0.005000

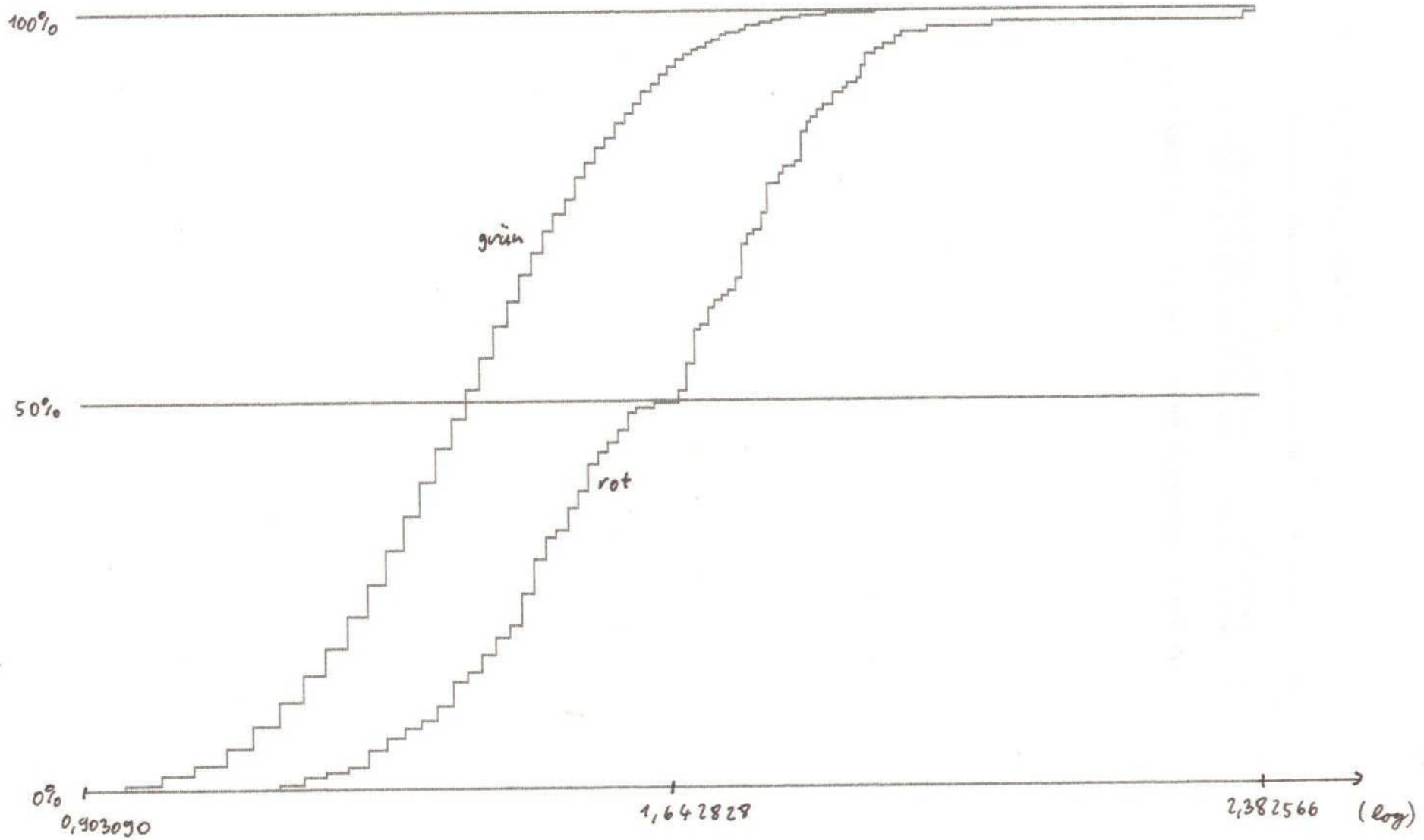
'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
94.96%	5.04%	99.86%	0.14%	2.19%	46.67%	1.900000
95.68%	4.32%	99.88%	0.12%	2.21%	46.15%	1.905000
95.68%	4.32%	99.88%	0.12%	2.21%	46.15%	1.910000
95.68%	4.32%	99.90%	0.10%	2.21%	50.00%	1.915000
96.40%	3.60%	99.90%	0.10%	2.22%	45.45%	1.920000
97.12%	2.88%	99.93%	0.07%	2.24%	50.00%	1.925000
97.12%	2.88%	99.93%	0.07%	2.24%	50.00%	1.930000
97.12%	2.88%	99.93%	0.07%	2.24%	50.00%	1.935000
97.12%	2.88%	99.97%	0.03%	2.24%	66.67%	1.940000
97.12%	2.88%	99.97%	0.03%	2.24%	66.67%	1.945000
97.12%	2.88%	99.97%	0.03%	2.24%	66.67%	1.950000
97.12%	2.88%	99.97%	0.03%	2.24%	66.67%	1.955000
97.84%	2.16%	99.97%	0.03%	2.25%	60.00%	1.960000
97.84%	2.16%	99.97%	0.03%	2.25%	60.00%	1.965000

97.84%	2.16%	99.97%	0.03%	2.25%	60.00%	1.970000
97.84%	2.16%	99.97%	0.03%	2.25%	60.00%	1.975000
97.84%	2.16%	99.97%	0.03%	2.25%	60.00%	1.980000
97.84%	2.16%	99.97%	0.03%	2.25%	60.00%	1.985000
97.84%	2.16%	99.97%	0.03%	2.25%	60.00%	1.990000
97.84%	2.16%	99.97%	0.03%	2.25%	60.00%	1.995000
97.84%	2.16%	99.97%	0.03%	2.25%	60.00%	2.000000

Anfangswert: 0.903090 Endwert: 1.300000
Intervallbreite: 0.019846

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
0.00%	100.00%	0.15%	99.85%	0.00%	2.31%	0.903090
0.00%	100.00%	0.15%	99.85%	0.00%	2.31%	0.922935
0.00%	100.00%	0.15%	99.85%	0.00%	2.31%	0.942781
0.00%	100.00%	0.63%	99.37%	0.00%	2.32%	0.962626
0.00%	100.00%	0.63%	99.37%	0.00%	2.32%	0.982472
0.00%	100.00%	1.78%	98.22%	0.00%	2.34%	1.002317
0.00%	100.00%	1.78%	98.22%	0.00%	2.34%	1.022163
0.00%	100.00%	3.10%	96.90%	0.00%	2.37%	1.042008
0.00%	100.00%	3.10%	96.90%	0.00%	2.37%	1.061854
0.00%	100.00%	5.39%	94.61%	0.00%	2.43%	1.081699
0.00%	100.00%	5.39%	94.61%	0.00%	2.43%	1.101545
0.00%	100.00%	8.07%	91.93%	0.00%	2.50%	1.121390
0.00%	100.00%	8.07%	91.93%	0.00%	2.50%	1.141236
0.72%	99.28%	11.20%	88.80%	0.15%	2.57%	1.161081
1.44%	98.56%	14.64%	85.36%	0.23%	2.65%	1.180927
1.44%	98.56%	14.64%	85.36%	0.23%	2.65%	1.200772
2.16%	97.84%	18.19%	81.81%	0.28%	2.74%	1.220618
2.88%	97.12%	22.15%	77.85%	0.31%	2.86%	1.240463
5.04%	94.96%	26.46%	73.54%	0.45%	2.95%	1.260309
6.47%	93.53%	30.71%	69.29%	0.49%	3.08%	1.280154
6.47%	93.53%	30.71%	69.29%	0.49%	3.08%	1.300000


457
12
0,903090
1,642828
2,382566
(log)
Maximum



Maximum

6.3.1.2.Mittlere Intensität

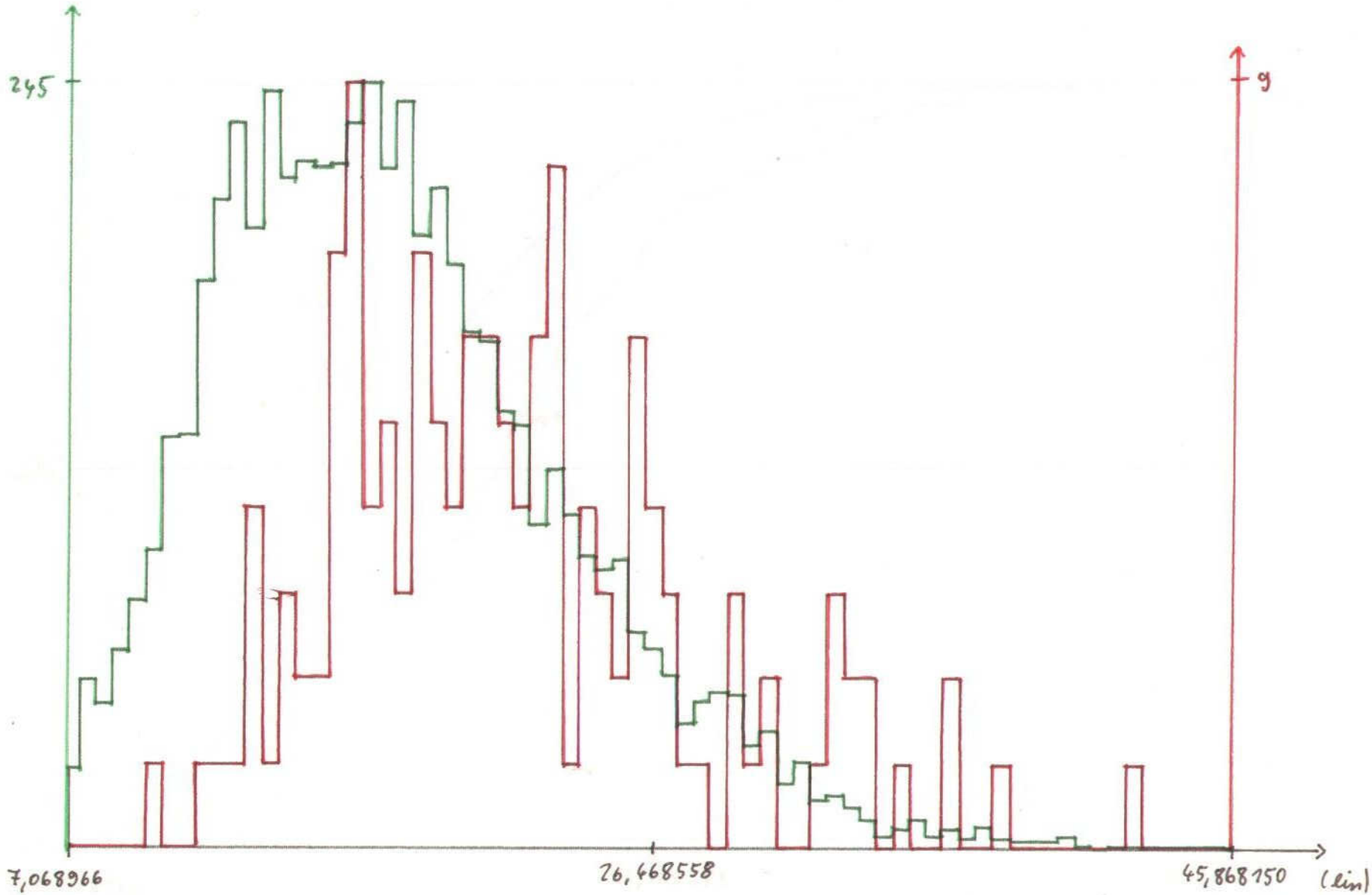
Eine Extravasations-Erkennungsgrenze kann hier nicht ermittelt werden.

17.81% der Nicht-Extravasationen besitzen einen mittleren Grauwert kleiner oder gleich 12.613793. Nur 2.16% aller Extravasationen fallen in diesen Bereich.

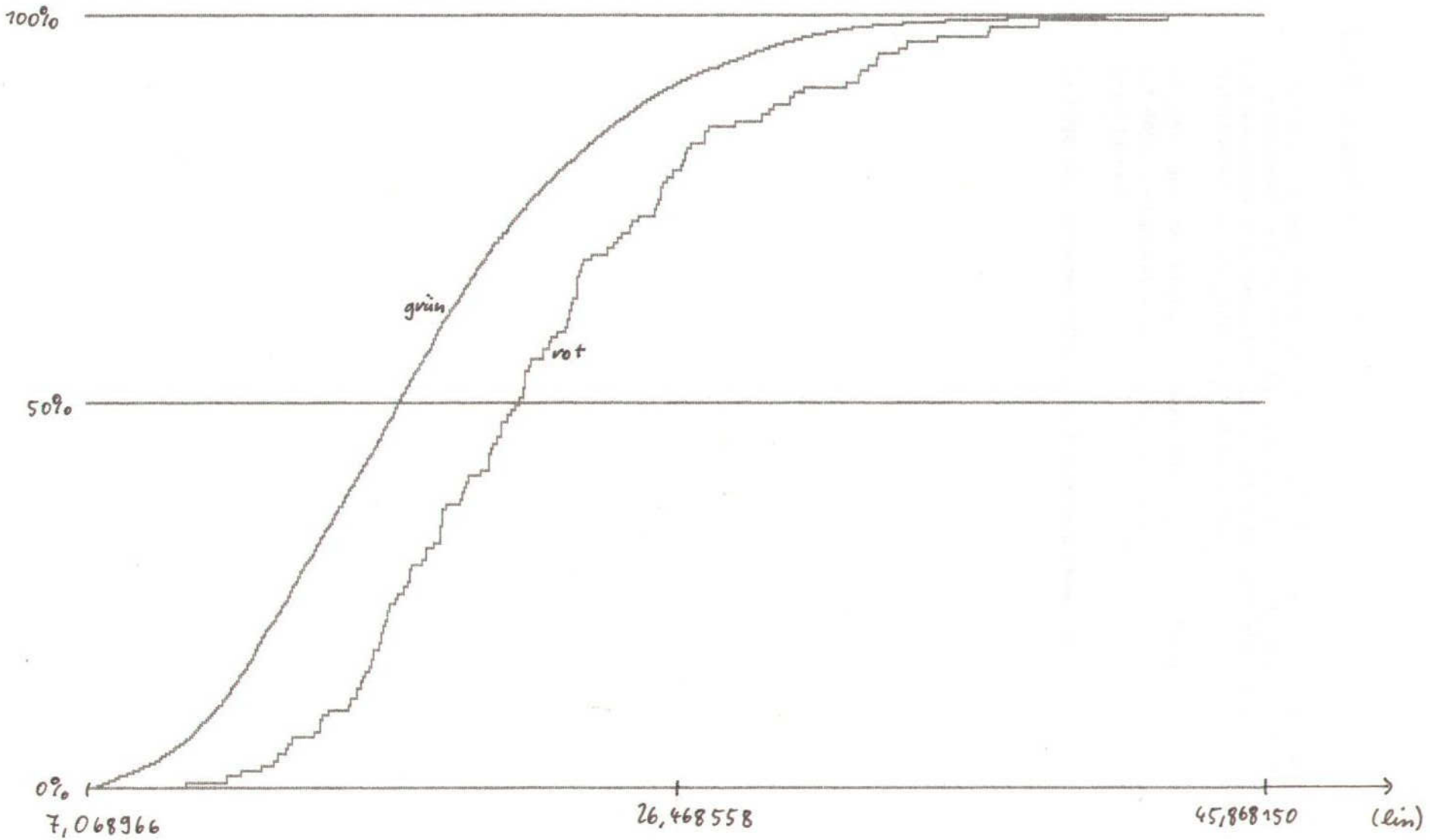
Die folgenden vier Seiten enthalten die Auswertung dieser Größe.

```
Datenbezeichnung: Mittelwert;   Anzahl der Daten 1: 139
Datenbezeichnung: Mittelwert;   Anzahl der Daten 2: 5900
70 Säulen       Lineare Darstellung
Reales Minimum: 7.068966        Reales Maximum: 45.321683
Minimum im Diagramm: 7.068966   Maximum im Diagramm: 45.868150
Intervallbreite: 0.546467
Histogramm 1 (rot)    : Maximum = 9
Histogramm 2 (gruen) : Maximum = 245
Angeglichene Histogramme


Anfangswert: 7.068966   Endwert: 45.321683
Intervallbreite: 1.912636

'1'<=   '1'>    '2'<=   '2'>    'p1'<=  'p1'>   als
0.00%   100.00% 0.02%   99.98%  0.00%   2.30%   7.068966
0.00%   100.00% 2.68%   97.32%  0.00%   2.36%   8.981602
0.72%   99.28%  8.36%   91.64%  0.20%   2.49%   10.894238
2.88%   97.12%  19.37%  80.63%  0.35%   2.76%   12.806874
8.63%   91.37%  32.19%  67.81%  0.63%   3.08%   14.719509
18.71%  81.29%  45.17%  54.83%  0.97%   3.38%   16.632145
31.65%  68.35%  59.14%  40.86%  1.25%   3.79%   18.544781
45.32%  54.68%  70.68%  29.32%  1.49%   4.21%   20.457417
58.27%  41.73%  79.41%  20.59%  1.70%   4.56%   22.370053
70.50%  29.50%  85.88%  14.12%  1.90%   4.69%   24.282689
79.86%  20.14%  90.86%  9.14%   2.03%   4.94%   26.195325
85.61%  14.39%  94.02%  5.98%   2.10%   5.36%   28.107960
89.21%  10.79%  96.59%  3.41%   2.13%   6.94%   30.020596
91.37%  8.63%   98.22%  1.78%   2.14%   10.26%  31.933232
96.40%  3.60%   98.98%  1.02%   2.24%   7.69%   33.845868
97.12%  2.88%   99.36%  0.64%   2.25%   9.52%   35.758504
98.56%  1.44%   99.64%  0.36%   2.28%   8.70%   37.671140
99.28%  0.72%   99.78%  0.22%   2.29%   7.14%   39.583775
99.28%  0.72%   99.88%  0.12%   2.29%   12.50%  41.496411
100.00% 0.00%   99.95%  0.05%   2.30%   0.00%   43.409047
100.00% 0.00%   100.00% 0.00%   2.30%   999.00% 45.321683


Anfangswert: 7.068966   Endwert: 14.000000
Intervallbreite: 0.346552

'1'<=   '1'>    '2'<=   '2'>    'p1'<=  'p1'>   als
0.00%   100.00% 0.02%   99.98%  0.00%   2.30%   7.068966
0.00%   100.00% 0.17%   99.83%  0.00%   2.31%   7.415518
0.00%   100.00% 0.66%   99.34%  0.00%   2.32%   7.762069
0.00%   100.00% 1.27%   98.73%  0.00%   2.33%   8.108621
0.00%   100.00% 1.76%   98.24%  0.00%   2.34%   8.455173
0.00%   100.00% 2.36%   97.64%  0.00%   2.36%   8.801724
0.00%   100.00% 2.98%   97.02%  0.00%   2.37%   9.148276
0.00%   100.00% 3.75%   96.25%  0.00%   2.39%   9.494828
0.00%   100.00% 4.59%   95.41%  0.00%   2.41%   9.841380
0.00%   100.00% 5.63%   94.37%  0.00%   2.44%   10.187931
0.72%   99.28%  6.86%   93.14%  0.25%   2.45%   10.534483
0.72%   99.28%  8.29%   91.71%  0.20%   2.49%   10.881035
0.72%   99.28%  9.59%   90.41%  0.18%   2.52%   11.227586
0.72%   99.28%  11.34%  88.66%  0.15%   2.57%   11.574138
```

1.44%	98.56%	13.29%	86.71%	0.25%	2.61%	11.920690
2.16%	97.84%	15.39%	84.61%	0.33%	2.65%	12.267241
2.16%	97.84%	17.81%	82.19%	0.28%	2.73%	12.613793
2.88%	97.12%	20.42%	79.58%	0.33%	2.80%	12.960345
3.60%	96.40%	22.37%	77.63%	0.38%	2.84%	13.306897
5.76%	94.24%	24.66%	75.34%	0.55%	2.86%	13.653448
6.47%	93.53%	27.24%	72.76%	0.56%	2.94%	14.000000



Mittelwert



Mittelwert

6.3.1.3.Fläche

Regionen mit einer Fläche größer als $10^{3.15}$(*100µm²) = 1412.54(*100µm²) = 0.141254mm² sind zu 50% Extravasationen. Dies ist der Fall für 45.32% aller Extravasationen, d.h. dieses Kriterium kann fast die Hälfte aller Extravasationen mit einer Wahrscheinlichkeit von größer gleich 50% erkennen.

14.32% der Nicht-Extravasationen besitzen eine Fläche kleiner oder gleich $10^{1.486895}$(*100µm²) = 30.68(*100µm²). Nur 2.16% aller Extravasationen fallen in diesen Bereich.

Die folgenden vier Seiten enthalten die Auswertung dieser Größe.

Datenbezeichnung: Flaeche; Anzahl der Daten 1: 139
Datenbezeichnung: Flaeche; Anzahl der Daten 2: 5900
70 Säulen Logarithmische Darstellung
Reales Minimum: 1.399069 Reales Maximum: 4.751771
Minimum im Diagramm: 1.399069 Maximum im Diagramm: 4.799667
Intervallbreite: 0.047896
Histogramm 1 (rot) : Maximum = 5
Histogramm 2 (gruen) : Maximum = 593
Angeglichene Histogramme

Anfangswert: 1.399069 Endwert: 4.751771
Intervallbreite: 0.167635

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
1.44%	98.56%	2.58%	97.42%	1.30%	2.33%	1.399069
5.04%	94.96%	25.19%	74.81%	0.47%	2.90%	1.566704
8.63%	91.37%	44.19%	55.81%	0.46%	3.71%	1.734339
12.23%	87.77%	59.63%	40.37%	0.48%	4.87%	1.901974
19.42%	80.58%	70.85%	29.15%	0.64%	6.11%	2.069609
24.46%	75.54%	79.80%	20.20%	0.72%	8.10%	2.237244
33.09%	66.91%	86.81%	13.19%	0.89%	10.68%	2.404880
37.41%	62.59%	92.02%	7.98%	0.95%	15.59%	2.572515
41.73%	58.27%	95.54%	4.46%	1.02%	23.55%	2.740150
45.32%	54.68%	97.39%	2.61%	1.08%	33.04%	2.907785
51.80%	48.20%	98.58%	1.42%	1.22%	44.37%	3.075420
57.55%	42.45%	99.29%	0.71%	1.35%	58.42%	3.243055
66.19%	33.81%	99.59%	0.41%	1.54%	66.20%	3.410690
72.66%	27.34%	99.78%	0.22%	1.69%	74.51%	3.578325
81.29%	18.71%	99.90%	0.10%	1.88%	81.25%	3.745960
85.61%	14.39%	99.90%	0.10%	1.98%	76.92%	3.913595
89.21%	10.79%	99.93%	0.07%	2.06%	78.95%	4.081230
90.65%	9.35%	99.97%	0.03%	2.09%	86.67%	4.248866
92.81%	7.19%	100.00%	0.00%	2.14%	100.00%	4.416501
94.96%	5.04%	100.00%	0.00%	2.19%	100.00%	4.584136
100.00%	0.00%	100.00%	0.00%	2.30%	999.00%	4.751771

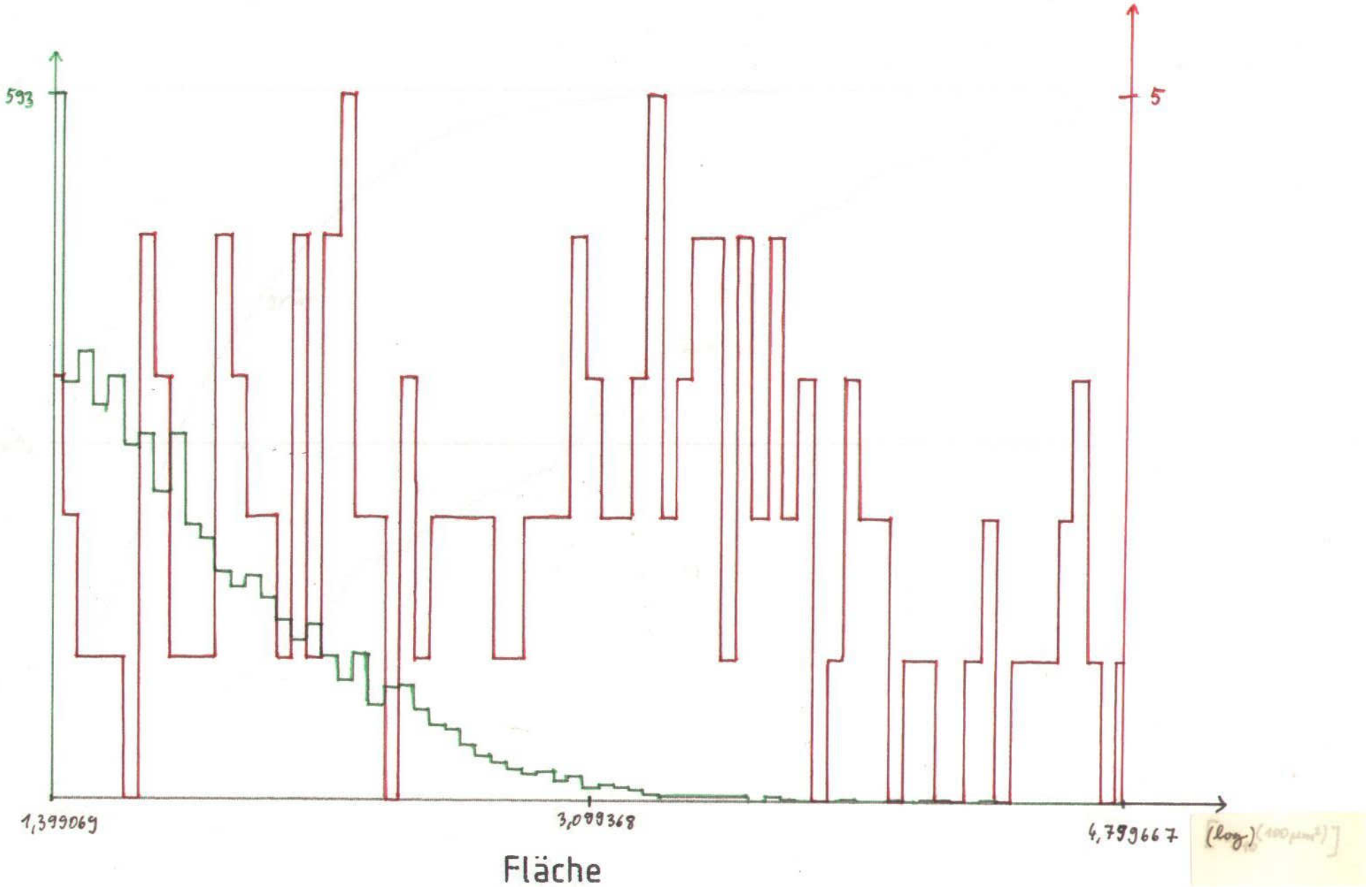
Anfangswert: 3.100000 Endwert: 3.200000
Intervallbreite: 0.005000

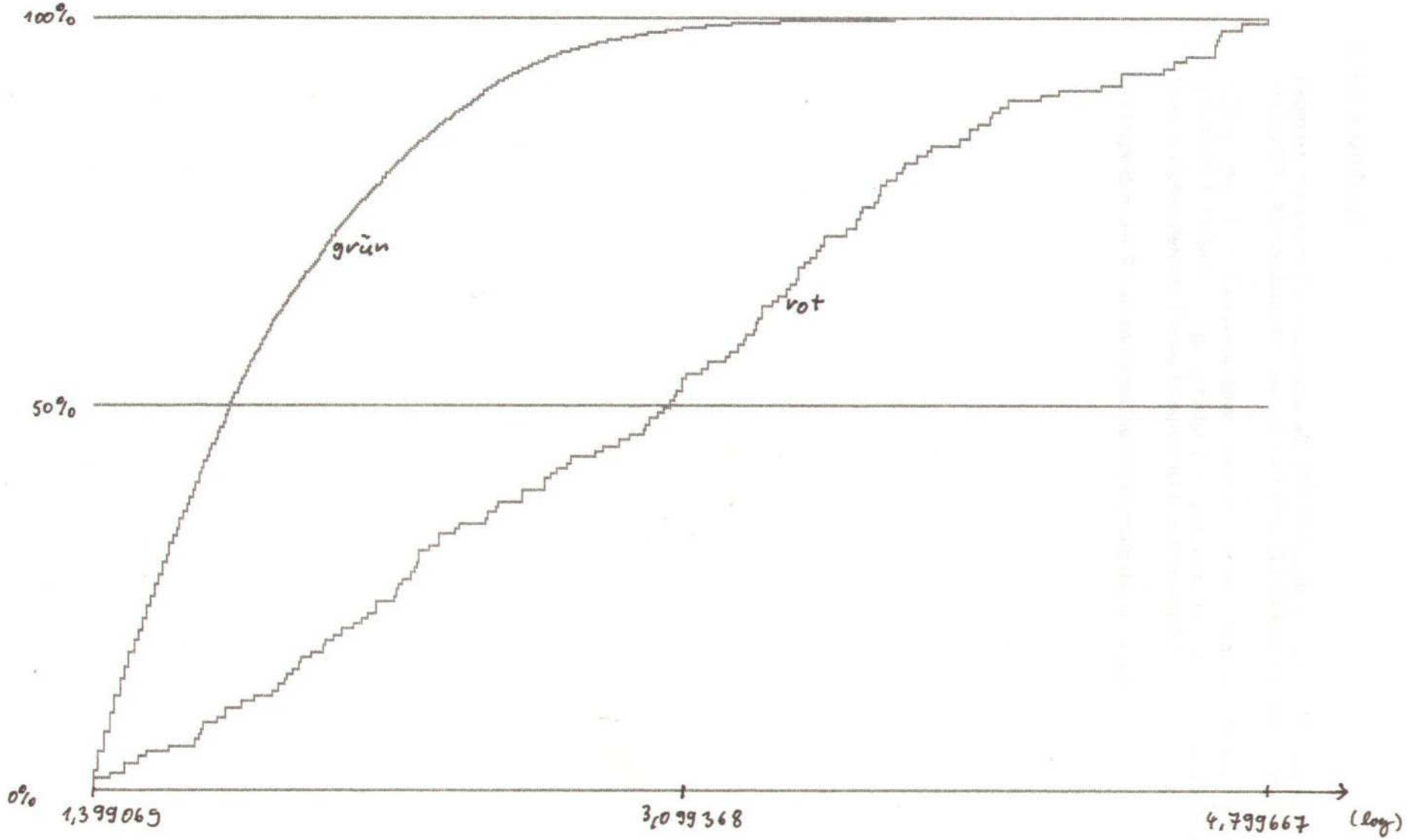
'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
53.96%	46.04%	98.69%	1.31%	1.27%	45.39%	3.100000
53.96%	46.04%	98.75%	1.25%	1.27%	46.38%	3.105000
53.96%	46.04%	98.75%	1.25%	1.27%	46.38%	3.110000
53.96%	46.04%	98.75%	1.25%	1.27%	46.38%	3.115000
53.96%	46.04%	98.75%	1.25%	1.27%	46.38%	3.120000
53.96%	46.04%	98.80%	1.20%	1.27%	47.41%	3.125000
53.96%	46.04%	98.86%	1.14%	1.27%	48.85%	3.130000
53.96%	46.04%	98.86%	1.14%	1.27%	48.85%	3.135000
54.68%	45.32%	98.86%	1.14%	1.29%	48.46%	3.140000
54.68%	45.32%	98.90%	1.10%	1.29%	49.22%	3.145000
54.68%	45.32%	98.93%	1.07%	1.29%	50.00%	3.150000
55.40%	44.60%	98.97%	1.03%	1.30%	50.41%	3.155000
55.40%	44.60%	98.97%	1.03%	1.30%	50.41%	3.160000
55.40%	44.60%	98.98%	1.02%	1.30%	50.82%	3.165000

55.40%	44.60%	99.00%	1.00%	1.30%	51.24%	3.170000
55.40%	44.60%	99.00%	1.00%	1.30%	51.24%	3.175000
55.40%	44.60%	99.00%	1.00%	1.30%	51.24%	3.180000
55.40%	44.60%	99.03%	0.97%	1.30%	52.10%	3.185000
55.40%	44.60%	99.07%	0.93%	1.30%	52.99%	3.190000
55.40%	44.60%	99.08%	0.92%	1.30%	53.45%	3.195000
55.40%	44.60%	99.08%	0.92%	1.30%	53.45%	3.200000

Anfangswert: 1.399069 Endwert: 1.650000
Intervallbreite: 0.012547

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
1.44%	98.56%	2.58%	97.42%	1.30%	2.33%	1.399069
1.44%	98.56%	2.58%	97.42%	1.30%	2.33%	1.411616
1.44%	98.56%	5.12%	94.88%	0.66%	2.39%	1.424162
1.44%	98.56%	7.58%	92.42%	0.45%	2.45%	1.436709
2.16%	97.84%	10.05%	89.95%	0.50%	2.50%	1.449255
2.16%	97.84%	12.36%	87.64%	0.41%	2.56%	1.461802
2.16%	97.84%	12.36%	87.64%	0.41%	2.56%	1.474348
2.16%	97.84%	14.32%	85.68%	0.35%	2.62%	1.486895
3.60%	96.40%	15.97%	84.03%	0.53%	2.63%	1.499441
3.60%	96.40%	17.71%	82.29%	0.48%	2.69%	1.511988
3.60%	96.40%	19.37%	80.63%	0.44%	2.74%	1.524534
4.32%	95.68%	20.81%	79.19%	0.49%	2.77%	1.537081
4.32%	95.68%	22.34%	77.66%	0.45%	2.82%	1.549628
5.04%	94.96%	23.83%	76.17%	0.50%	2.85%	1.562174
5.04%	94.96%	25.19%	74.81%	0.47%	2.90%	1.574721
5.04%	94.96%	27.93%	72.07%	0.42%	3.01%	1.587267
5.04%	94.96%	29.36%	70.64%	0.40%	3.07%	1.599814
5.04%	94.96%	30.49%	69.51%	0.39%	3.12%	1.612360
5.76%	94.24%	31.88%	68.12%	0.42%	3.16%	1.624907
5.76%	94.24%	33.95%	66.05%	0.40%	3.25%	1.637453
5.76%	94.24%	35.07%	64.93%	0.39%	3.31%	1.650000


593
5
1,399069
3,099368
4,799667
(log) [100 µm²]
Fläche



Fläche

6.3.2.Umfang

Regionen mit einem Umfang größer als $10^{2.8}$(*10µm) = 630.96(*10µm) = 6.3096mm sind zu 50% Extravasationen. Dies ist der Fall für 33.09% aller Extravasationen.

5.39% der Nicht-Extravasationen besitzen einen Umfang kleiner oder gleich $10^{1.430855(*10\mu m)}$ = 26.97(*10µm) = 269.7µm. Nur 2.16% aller Extravasationen fallen in diesen Bereich. Dieses Kriterium ist hier nicht ergiebig.

Die folgenden vier Seiten enthalten die Auswertung dieser Größe.

Datenbezeichnung: Umfang; Anzahl der Daten 1: 139
Datenbezeichnung: Umfang; Anzahl der Daten 2: 5900
70 Säulen Logarithmische Darstellung
Reales Minimum: 1.333373 Reales Maximum: 3.973242
Minimum im Diagramm: 1.333373 Maximum im Diagramm: 4.010954
Intervallbreite: 0.037712
Histogramm 1 (rot) : Maximum = 6
Histogramm 2 (gruen) : Maximum = 460
Angeglichene Histogramme

Anfangswert: 1.333373 Endwert: 3.973242
Intervallbreite: 0.131993

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
0.00%	100.00%	0.02%	99.98%	0.00%	2.30%	1.333373
4.32%	95.68%	11.54%	88.46%	0.87%	2.49%	1.465367
6.47%	93.53%	35.54%	64.46%	0.43%	3.31%	1.597360
13.67%	86.33%	54.69%	45.31%	0.59%	4.30%	1.729353
20.14%	79.86%	68.03%	31.97%	0.69%	5.56%	1.861347
28.06%	71.94%	78.61%	21.39%	0.83%	7.34%	1.993340
33.81%	66.19%	86.46%	13.54%	0.91%	10.33%	2.125334
37.41%	62.59%	91.68%	8.32%	0.95%	15.05%	2.257327
43.17%	56.83%	95.03%	4.97%	1.06%	21.24%	2.389321
48.20%	51.80%	97.41%	2.59%	1.15%	32.00%	2.521314
56.83%	43.17%	98.49%	1.51%	1.34%	40.27%	2.653308
66.91%	33.09%	99.19%	0.81%	1.56%	48.94%	2.785301
74.82%	25.18%	99.53%	0.47%	1.74%	55.56%	2.917294
76.98%	23.02%	99.80%	0.20%	1.78%	72.73%	3.049288
82.01%	17.99%	99.83%	0.17%	1.90%	71.43%	3.181281
86.33%	13.67%	99.88%	0.12%	2.00%	73.08%	3.313275
89.21%	10.79%	99.93%	0.07%	2.06%	78.95%	3.445268
90.65%	9.35%	99.95%	0.05%	2.09%	81.25%	3.577262
92.09%	7.91%	99.95%	0.05%	2.12%	78.57%	3.709255
96.40%	3.60%	99.98%	0.02%	2.22%	83.33%	3.841248
100.00%	0.00%	100.00%	0.00%	2.30%	999.00%	3.973242

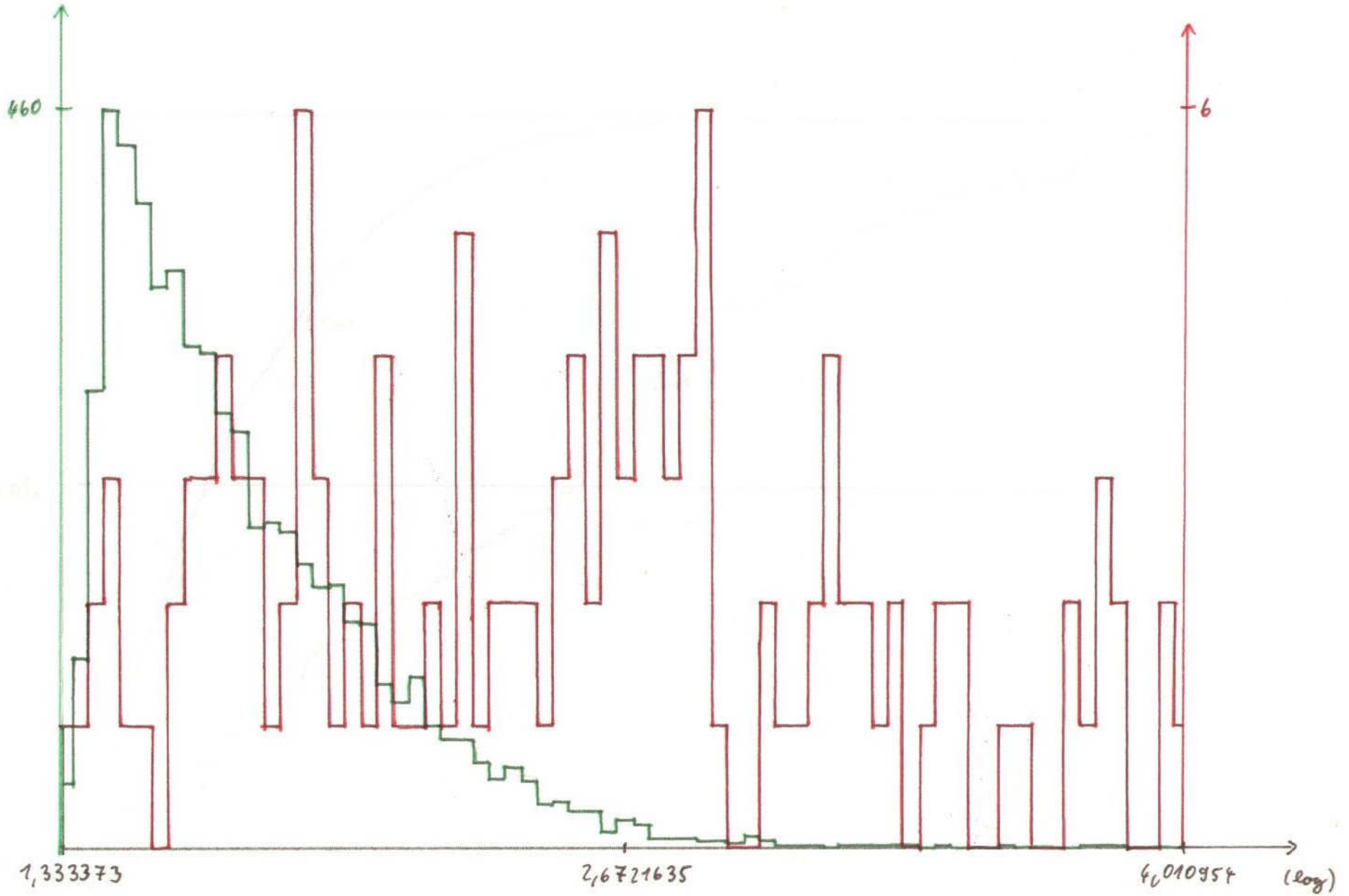
Anfangswert: 2.700000 Endwert: 2.900000
Intervallbreite: 0.010000

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
60.43%	39.57%	98.85%	1.15%	1.42%	44.72%	2.700000
60.43%	39.57%	98.95%	1.05%	1.42%	47.01%	2.710000
61.87%	38.13%	99.02%	0.98%	1.45%	47.75%	2.720000
61.87%	38.13%	99.05%	0.95%	1.45%	48.62%	2.730000
62.59%	37.41%	99.10%	0.90%	1.47%	49.52%	2.740000
64.75%	35.25%	99.12%	0.88%	1.52%	48.51%	2.750000
64.75%	35.25%	99.15%	0.85%	1.52%	49.49%	2.760000
65.47%	34.53%	99.15%	0.85%	1.53%	48.98%	2.770000
66.19%	33.81%	99.17%	0.83%	1.55%	48.96%	2.780000
66.91%	33.09%	99.20%	0.80%	1.56%	49.46%	2.790000
66.91%	33.09%	99.22%	0.78%	1.56%	50.00%	2.800000
67.63%	32.37%	99.27%	0.73%	1.58%	51.14%	2.810000
67.63%	32.37%	99.29%	0.71%	1.58%	51.72%	2.820000
69.06%	30.94%	99.32%	0.68%	1.61%	51.81%	2.830000
69.78%	30.22%	99.34%	0.66%	1.63%	51.85%	2.840000

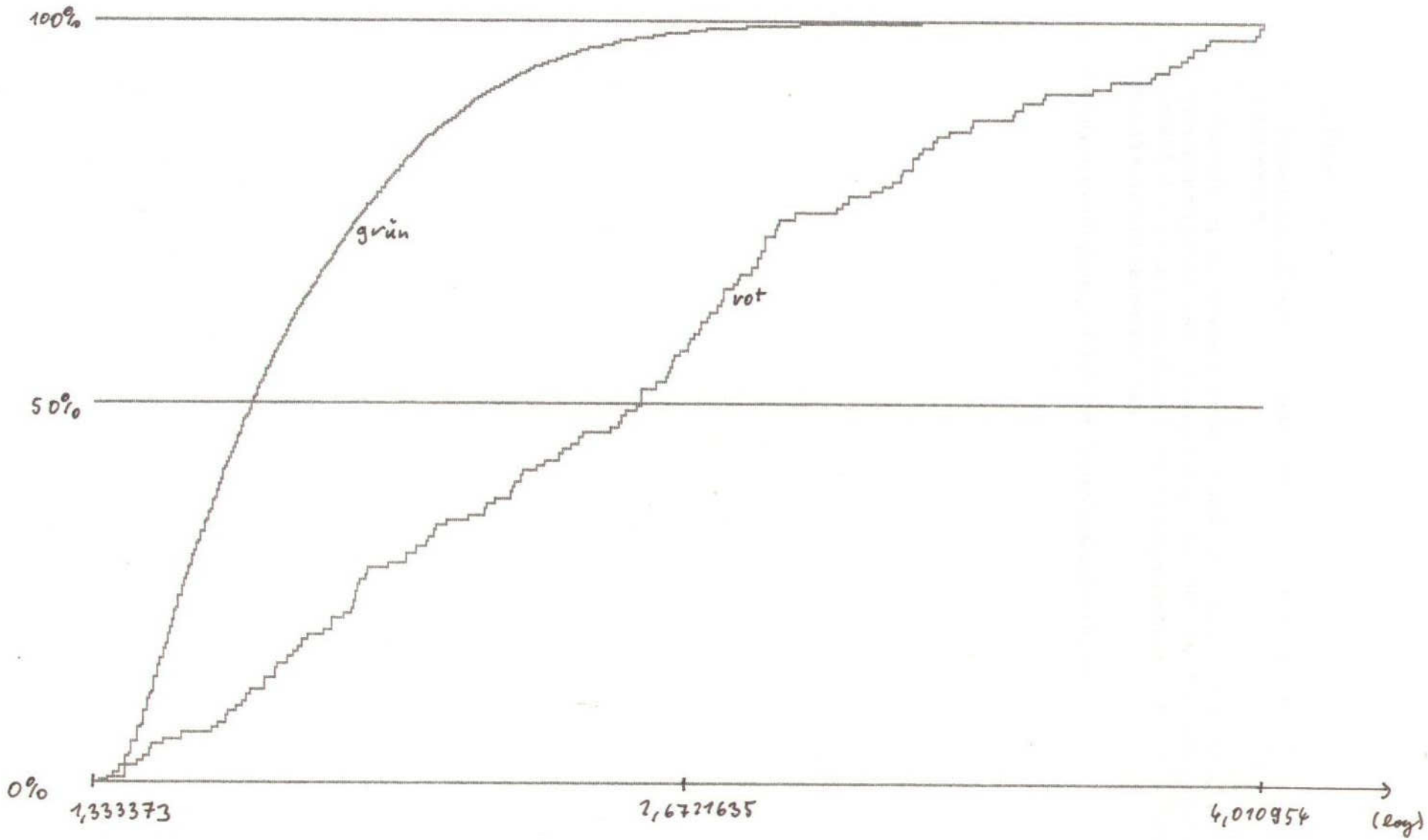
71.94%	28.06%	99.36%	0.64%	1.68%	50.65%	2.850000
71.94%	28.06%	99.39%	0.61%	1.68%	52.00%	2.860000
73.38%	26.62%	99.42%	0.58%	1.71%	52.11%	2.870000
74.10%	25.90%	99.44%	0.56%	1.73%	52.17%	2.880000
74.10%	25.90%	99.46%	0.54%	1.73%	52.94%	2.890000
74.10%	25.90%	99.47%	0.53%	1.72%	53.73%	2.900000

Anfangswert: 1.333373 Endwert: 1.550000
Intervallbreite: 0.010831

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
0.00%	100.00%	0.02%	99.98%	0.00%	2.30%	1.333373
0.00%	100.00%	0.03%	99.97%	0.00%	2.30%	1.344204
0.00%	100.00%	0.17%	99.83%	0.00%	2.31%	1.355036
0.00%	100.00%	0.39%	99.61%	0.00%	2.31%	1.365867
0.72%	99.28%	0.69%	99.31%	2.38%	2.30%	1.376699
0.72%	99.28%	1.39%	98.61%	1.20%	2.32%	1.387530
0.72%	99.28%	2.24%	97.76%	0.75%	2.34%	1.398361
1.44%	98.56%	2.69%	97.31%	1.24%	2.33%	1.409193
2.16%	97.84%	3.80%	96.20%	1.32%	2.34%	1.420024
2.16%	97.84%	5.39%	94.61%	0.93%	2.38%	1.430855
2.88%	97.12%	7.14%	92.86%	0.94%	2.40%	1.441687
3.60%	96.40%	9.34%	90.66%	0.90%	2.44%	1.452518
4.32%	95.68%	11.54%	88.46%	0.87%	2.49%	1.463349
5.04%	94.96%	13.69%	86.31%	0.86%	2.53%	1.474181
5.04%	94.96%	15.32%	84.68%	0.77%	2.57%	1.485012
5.76%	94.24%	17.68%	82.32%	0.76%	2.63%	1.495843
5.76%	94.24%	19.44%	80.56%	0.69%	2.68%	1.506675
5.76%	94.24%	22.08%	77.92%	0.61%	2.77%	1.517506
5.76%	94.24%	24.44%	75.56%	0.55%	2.85%	1.528337
6.47%	93.53%	26.46%	73.54%	0.57%	2.91%	1.539169
6.47%	93.53%	28.14%	71.86%	0.54%	2.97%	1.550000



Umfang



Umfang

6.3.2.1.Rundheit

Eine Extravasations-Erkennungsgrenze kann hier wie schon beim Mittelwert nicht ermittelt werden.

Die Rundheit ist als Unterscheidungsmerkmal auch bei der Erkennung von Nicht-Extravasationsregionen nicht aussagekräftig. So besitzen bei einer Grenze von $10^{0.023674} = 1.056$ 2.16% der Extravasationsregionen einen größeren Wert, aber auch nur 3.49% der Nicht-Extravasationen.

Die folgenden vier Seiten enthalten die Auswertung dieser Größe.

Datenbezeichnung: Rundheit; Anzahl der Daten 1: 139
Datenbezeichnung: Rundheit; Anzahl der Daten 2: 5900
70 Säulen Logarithmische Darstellung
Reales Minimum: -1.241694 Reales Maximum: 0.113719
Minimum im Diagramm: -1.241694 Maximum im Diagramm: 0.133083
Intervallbreite: 0.019363
Histogramm 1 (rot) : Maximum = 9
Histogramm 2 (gruen) : Maximum = 254
Angeglichene Histogramme

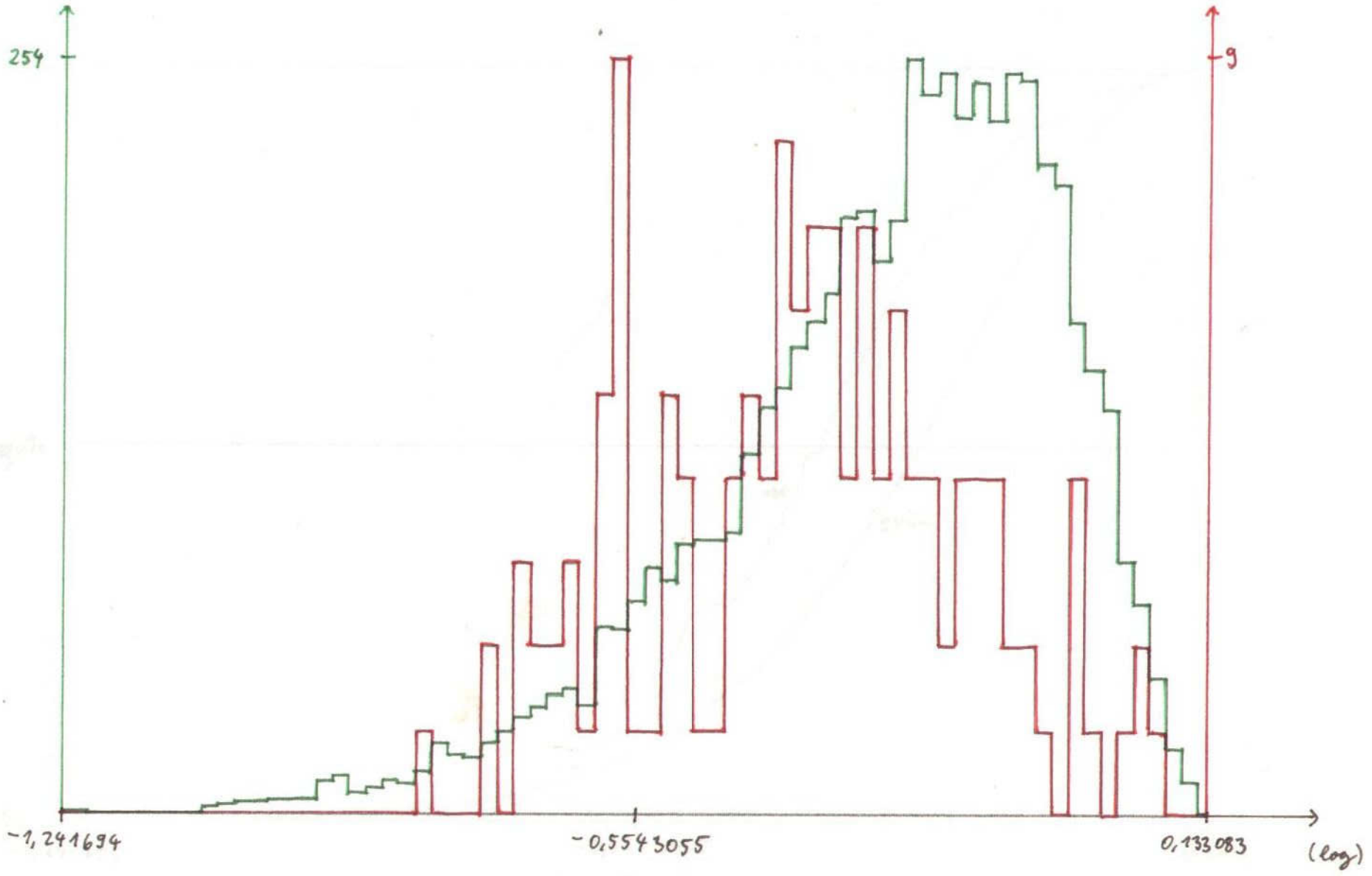
Anfangswert: -1.241694 Endwert: 0.113719
Intervallbreite: 0.067771

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
0.00%	100.00%	0.02%	99.98%	0.00%	2.30%	-1.241694
0.00%	100.00%	0.03%	99.97%	0.00%	2.30%	-1.173923
0.00%	100.00%	0.03%	99.97%	0.00%	2.30%	-1.106152
0.00%	100.00%	0.08%	99.92%	0.00%	2.30%	-1.038382
0.00%	100.00%	0.34%	99.66%	0.00%	2.31%	-0.970611
0.00%	100.00%	0.73%	99.27%	0.00%	2.32%	-0.902840
0.00%	100.00%	1.37%	98.63%	0.00%	2.33%	-0.835070
0.72%	99.28%	2.34%	97.66%	0.72%	2.34%	-0.767299
2.16%	97.84%	3.73%	96.27%	1.35%	2.34%	-0.699529
7.19%	92.81%	6.00%	94.00%	2.75%	2.27%	-0.631758
20.14%	79.86%	9.05%	90.95%	4.98%	2.03%	-0.563987
27.34%	72.66%	13.73%	86.27%	4.48%	1.95%	-0.496217
32.37%	67.63%	19.29%	80.71%	3.80%	1.94%	-0.428446
48.20%	51.80%	27.31%	72.69%	3.99%	1.65%	-0.360675
61.87%	38.13%	37.93%	62.07%	3.70%	1.43%	-0.292905
75.54%	24.46%	49.95%	50.05%	3.44%	1.14%	-0.225134
84.17%	15.83%	64.51%	35.49%	2.98%	1.04%	-0.157363
92.09%	7.91%	78.95%	21.05%	2.67%	0.88%	-0.089593
96.40%	3.60%	91.15%	8.85%	2.43%	0.95%	-0.021822
99.28%	0.72%	98.19%	1.81%	2.33%	0.93%	0.045949
100.00%	0.00%	99.98%	0.02%	2.30%	0.00%	0.113719

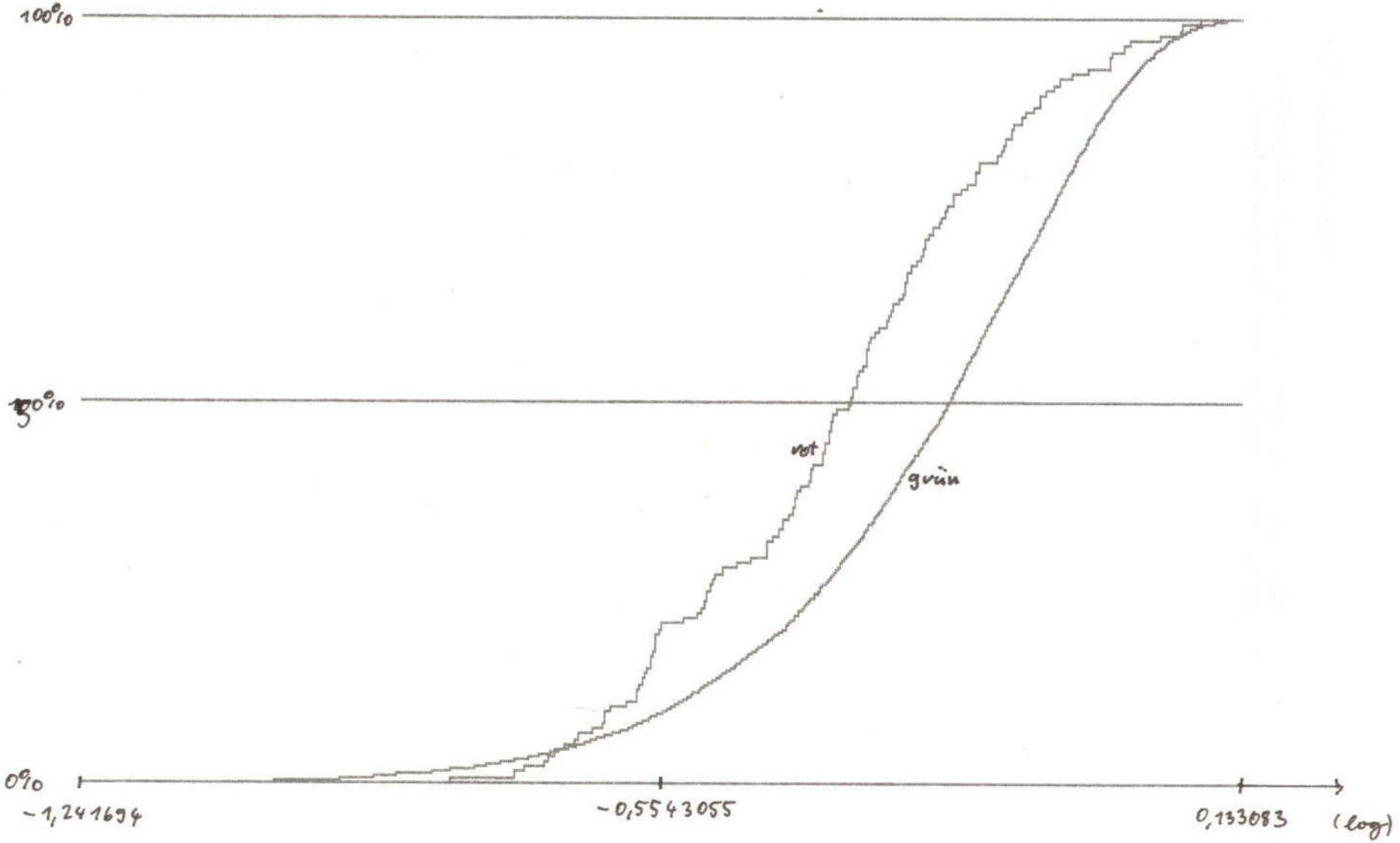
Anfangswert: -0.050000 Endwert: 0.113719
Intervallbreite: 0.008186

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
93.53%	6.47%	86.80%	13.20%	2.48%	1.14%	-0.050000
93.53%	6.47%	88.29%	11.71%	2.43%	1.29%	-0.041814
95.68%	4.32%	89.54%	10.46%	2.46%	0.96%	-0.033628
95.68%	4.32%	90.83%	9.17%	2.42%	1.10%	-0.025442
96.40%	3.60%	91.78%	8.22%	2.41%	1.02%	-0.017256
97.12%	2.88%	92.92%	7.08%	2.40%	0.95%	-0.009070
97.12%	2.88%	93.97%	6.03%	2.38%	1.11%	-0.000884
97.12%	2.88%	94.75%	5.25%	2.36%	1.27%	0.007302
97.12%	2.88%	95.83%	4.17%	2.33%	1.60%	0.015488
97.84%	2.16%	96.51%	3.49%	2.33%	1.44%	0.023674
97.84%	2.16%	97.24%	2.76%	2.32%	1.81%	0.031860
97.84%	2.16%	97.59%	2.41%	2.31%	2.07%	0.040046
99.28%	0.72%	98.27%	1.73%	2.32%	0.97%	0.048232
99.28%	0.72%	98.66%	1.34%	2.32%	1.25%	0.056418

99.28%	0.72%	99.00%	1.00%	2.31%	1.67%	0.064604
100.00%	0.00%	99.34%	0.66%	2.32%	0.00%	0.072790
100.00%	0.00%	99.53%	0.47%	2.31%	0.00%	0.080976
100.00%	0.00%	99.76%	0.24%	2.31%	0.00%	0.089162
100.00%	0.00%	99.88%	0.12%	2.30%	0.00%	0.097348
100.00%	0.00%	99.93%	0.07%	2.30%	0.00%	0.105533
100.00%	0.00%	100.00%	0.00%	2.30%	999.00%	0.113719



Rundheit



Rundheit

6.3.2.2.Länglichkeit

Die Länglichkeit kann als Unterscheidungskrirterium noch weniger eingesetzt werden, als die Rundheit. In den folgenden vier Seiten wird dieses deutlich.

Datenbezeichnung: Laenglichkeit; Anzahl der Daten 1: 139
Datenbezeichnung: Laenglichkeit; Anzahl der Daten 2: 5900
70 Säulen Logarithmische Darstellung
Reales Minimum: 0.012185 Reales Maximum: 2.427673
Minimum im Diagramm: 0.012185 Maximum im Diagramm: 2.462180
Intervallbreite: 0.034507
Histogramm 1 (rot) : Maximum = 8
Histogramm 2 (gruen) : Maximum = 256
Angeglichene Histogramme

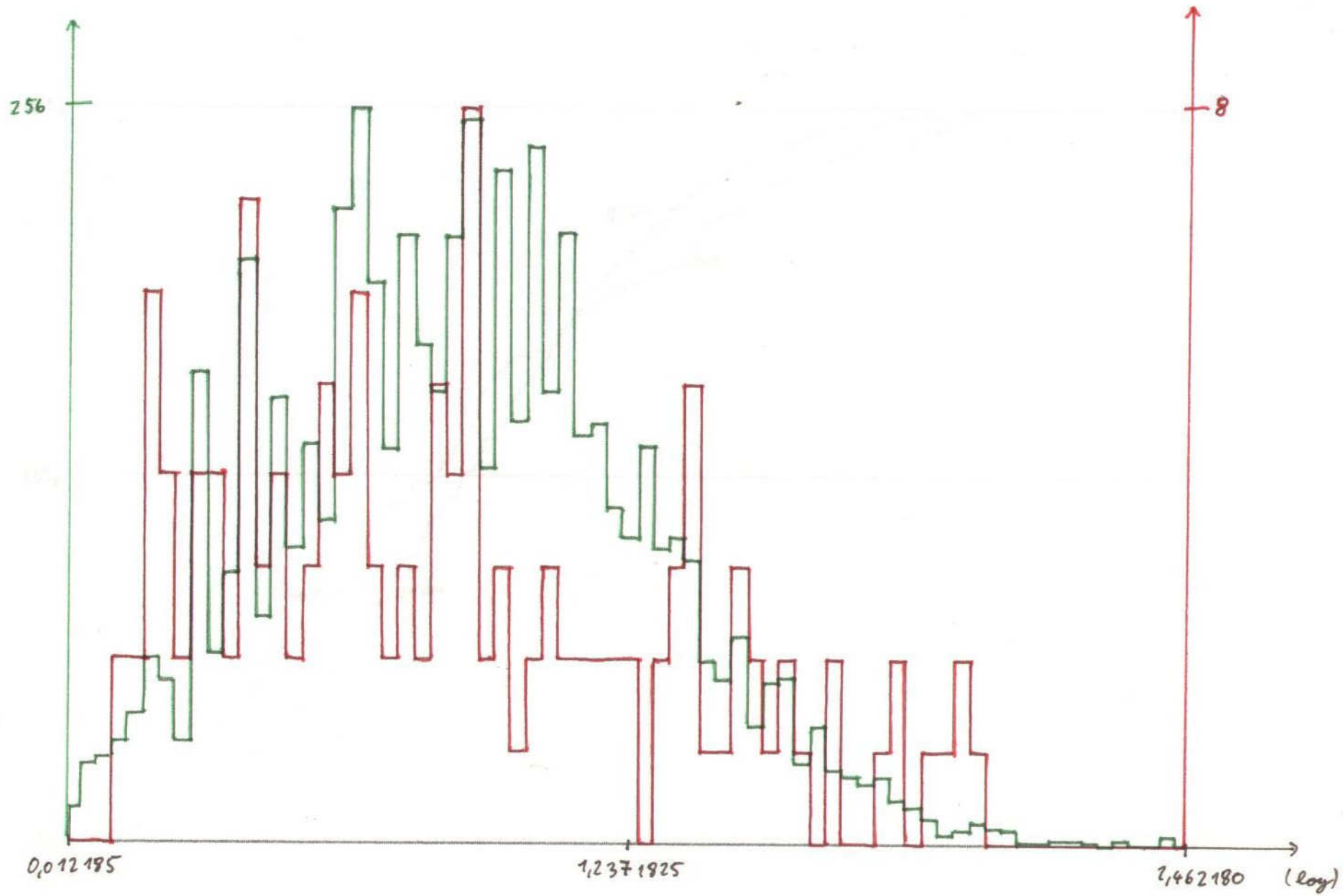
Anfangswert: 0.012185 Endwert: 2.427673
Intervallbreite: 0.120774

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
0.00%	100.00%	0.10%	99.90%	0.00%	2.30%	0.012185
0.72%	99.28%	1.53%	98.47%	1.10%	2.32%	0.132960
10.07%	89.93%	4.56%	95.44%	4.95%	2.17%	0.253734
17.27%	82.73%	9.88%	90.12%	3.95%	2.12%	0.374509
28.78%	71.22%	18.05%	81.95%	3.62%	2.01%	0.495283
38.85%	61.15%	27.07%	72.93%	3.27%	1.94%	0.616057
46.76%	53.24%	37.76%	62.24%	2.83%	1.98%	0.736832
56.12%	43.88%	50.17%	49.83%	2.57%	2.03%	0.857606
66.19%	33.81%	60.98%	39.02%	2.49%	2.00%	0.978381
71.94%	28.06%	72.03%	27.97%	2.30%	2.31%	1.099155
76.26%	23.74%	80.71%	19.29%	2.18%	2.82%	1.219929
80.58%	19.42%	87.39%	12.61%	2.13%	3.50%	1.340704
86.33%	13.67%	92.14%	7.86%	2.16%	3.93%	1.461478
91.37%	8.63%	95.44%	4.56%	2.21%	4.27%	1.582253
94.24%	5.76%	97.59%	2.41%	2.22%	5.33%	1.703027
94.96%	5.04%	98.90%	1.10%	2.21%	9.72%	1.823801
97.84%	2.16%	99.41%	0.59%	2.27%	7.89%	1.944576
100.00%	0.00%	99.76%	0.24%	2.31%	0.00%	2.065350
100.00%	0.00%	99.86%	0.14%	2.30%	0.00%	2.186125
100.00%	0.00%	99.93%	0.07%	2.30%	0.00%	2.306899
100.00%	0.00%	100.00%	0.00%	2.30%	999.00%	2.427673

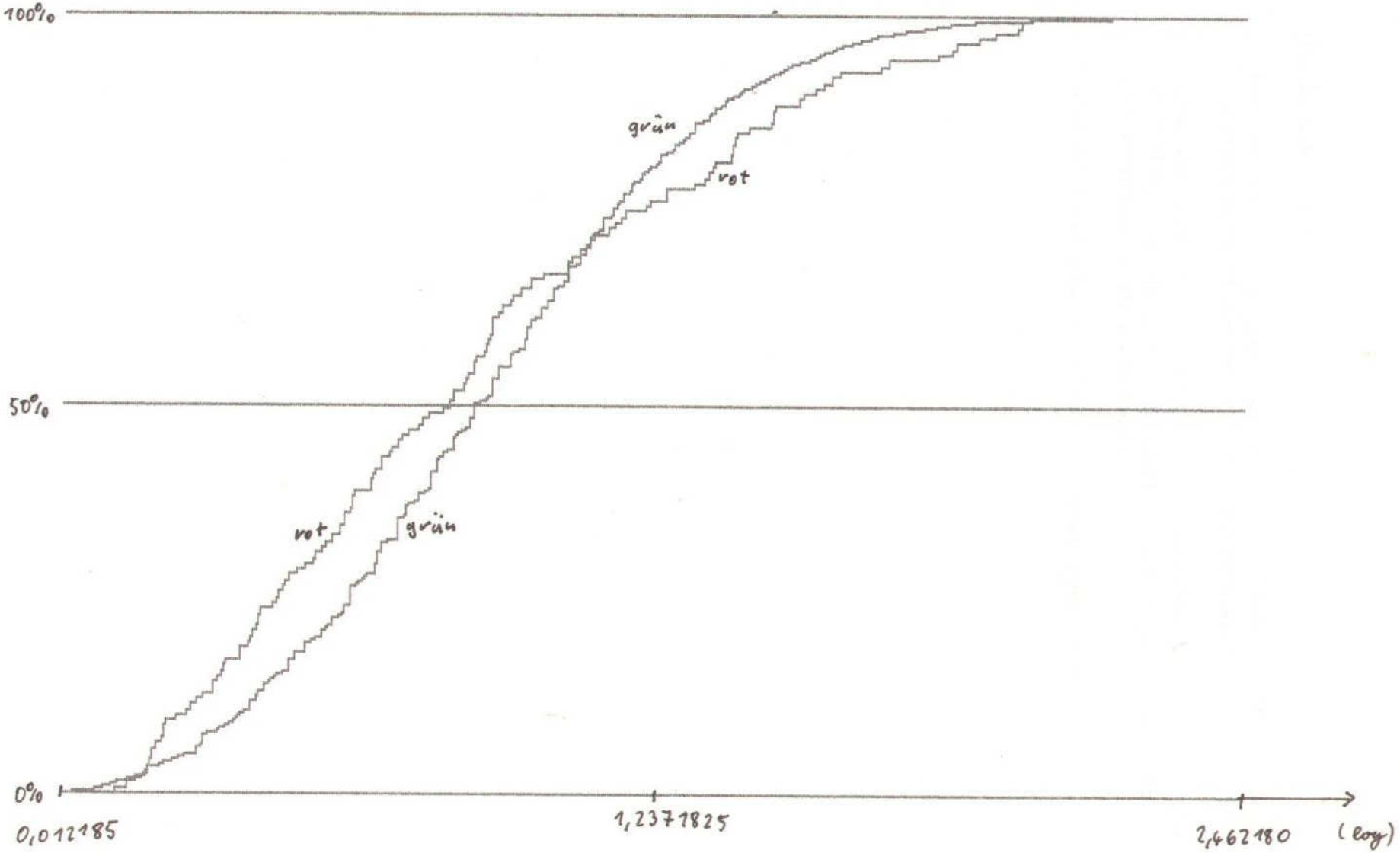
Anfangswert: 0.012185 Endwert: 0.250000
Intervallbreite: 0.011891

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
0.00%	100.00%	0.10%	99.90%	0.00%	2.30%	0.012185
0.00%	100.00%	0.15%	99.85%	0.00%	2.31%	0.024076
0.00%	100.00%	0.19%	99.81%	0.00%	2.31%	0.035967
0.00%	100.00%	0.20%	99.80%	0.00%	2.31%	0.047857
0.00%	100.00%	0.24%	99.76%	0.00%	2.31%	0.059748
0.00%	100.00%	0.42%	99.58%	0.00%	2.31%	0.071639
0.00%	100.00%	0.68%	99.32%	0.00%	2.32%	0.083530
0.00%	100.00%	0.71%	99.29%	0.00%	2.32%	0.095420
0.00%	100.00%	0.92%	99.08%	0.00%	2.32%	0.107311
0.00%	100.00%	1.19%	98.81%	0.00%	2.33%	0.119202
0.72%	99.28%	1.51%	98.49%	1.11%	2.32%	0.131093
0.72%	99.28%	1.61%	98.39%	1.04%	2.32%	0.142983
1.44%	98.56%	1.86%	98.14%	1.79%	2.31%	0.154874
2.16%	97.84%	1.95%	98.05%	2.54%	2.30%	0.166765

2.16%	97.84%	2.24%	97.76%	2.22%	2.30%	0.178656
3.60%	96.40%	3.32%	96.68%	2.49%	2.30%	0.190546
5.76%	94.24%	3.36%	96.64%	3.88%	2.25%	0.202437
6.47%	93.53%	3.61%	96.39%	4.05%	2.23%	0.214328
8.63%	91.37%	4.05%	95.95%	4.78%	2.19%	0.226219
9.35%	90.65%	4.25%	95.75%	4.92%	2.18%	0.238109
10.07%	89.93%	4.53%	95.47%	4.98%	2.17%	0.250000



Länglichkeit



Länglichkeit

6.3.2.3.Verwinkeltheit

Regionen mit einer Verwinkeltheit größer als $10^{2.44}$ = 275.42 sind zu 50% Extravasationen. Dies ist der Fall für 35.97% aller Extravasationen.

3.92% der Nicht-Extravasationen besitzen eine Verwinkeltheit kleiner oder gleich $10^{0.911499}$ = 8.156. 2.16% aller Extravasationen fallen in diesen Bereich. Dieses Kriterium ist hier nicht effektiv anwendbar.

Die folgenden vier Seiten enthalten die Auswertung dieser Größe.

Datenbezeichnung: Verwinkeltheit; Anzahl der Daten 1: 139
Datenbezeichnung: Verwinkeltheit; Anzahl der Daten 2: 5900
70 Säulen Logarithmische Darstellung
Reales Minimum: 0.798180 Reales Maximum: 3.751914
Minimum im Diagramm: 0.798180 Maximum im Diagramm: 3.794110
Intervallbreite: 0.042196
Histogramm 1 (rot) : Maximum = 6
Histogramm 2 (gruen) : Maximum = 492
Angeglichene Histogramme

Anfangswert: 0.798180 Endwert: 3.751914
Intervallbreite: 0.147687

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
0.72%	99.28%	1.64%	98.36%	1.02%	2.32%	0.798180
5.04%	94.96%	8.42%	91.58%	1.39%	2.38%	0.945867
7.91%	92.09%	29.29%	70.71%	0.63%	2.98%	1.093553
13.67%	86.33%	52.20%	47.80%	0.61%	4.08%	1.241240
20.14%	79.86%	68.44%	31.56%	0.69%	5.63%	1.388927
30.94%	69.06%	80.44%	19.56%	0.90%	7.68%	1.536613
35.25%	64.75%	88.24%	11.76%	0.93%	11.48%	1.684300
38.13%	61.87%	92.78%	7.22%	0.96%	16.80%	1.831987
45.32%	54.68%	95.73%	4.27%	1.10%	23.17%	1.979673
49.64%	50.36%	97.34%	2.66%	1.19%	30.84%	2.127360
57.55%	42.45%	98.39%	1.61%	1.36%	38.31%	2.275047
64.03%	35.97%	99.10%	0.90%	1.50%	48.54%	2.422733
68.35%	31.65%	99.39%	0.61%	1.59%	55.00%	2.570420
75.54%	24.46%	99.61%	0.39%	1.76%	59.65%	2.718107
77.70%	22.30%	99.78%	0.22%	1.80%	70.45%	2.865794
84.89%	15.11%	99.88%	0.12%	1.96%	75.00%	3.013480
87.77%	12.23%	99.92%	0.08%	2.03%	77.27%	3.161167
90.65%	9.35%	99.93%	0.07%	2.09%	76.47%	3.308854
92.09%	7.91%	99.95%	0.05%	2.12%	78.57%	3.456540
97.12%	2.88%	99.97%	0.03%	2.24%	66.67%	3.604227
100.00%	0.00%	100.00%	0.00%	2.30%	999.00%	3.751914

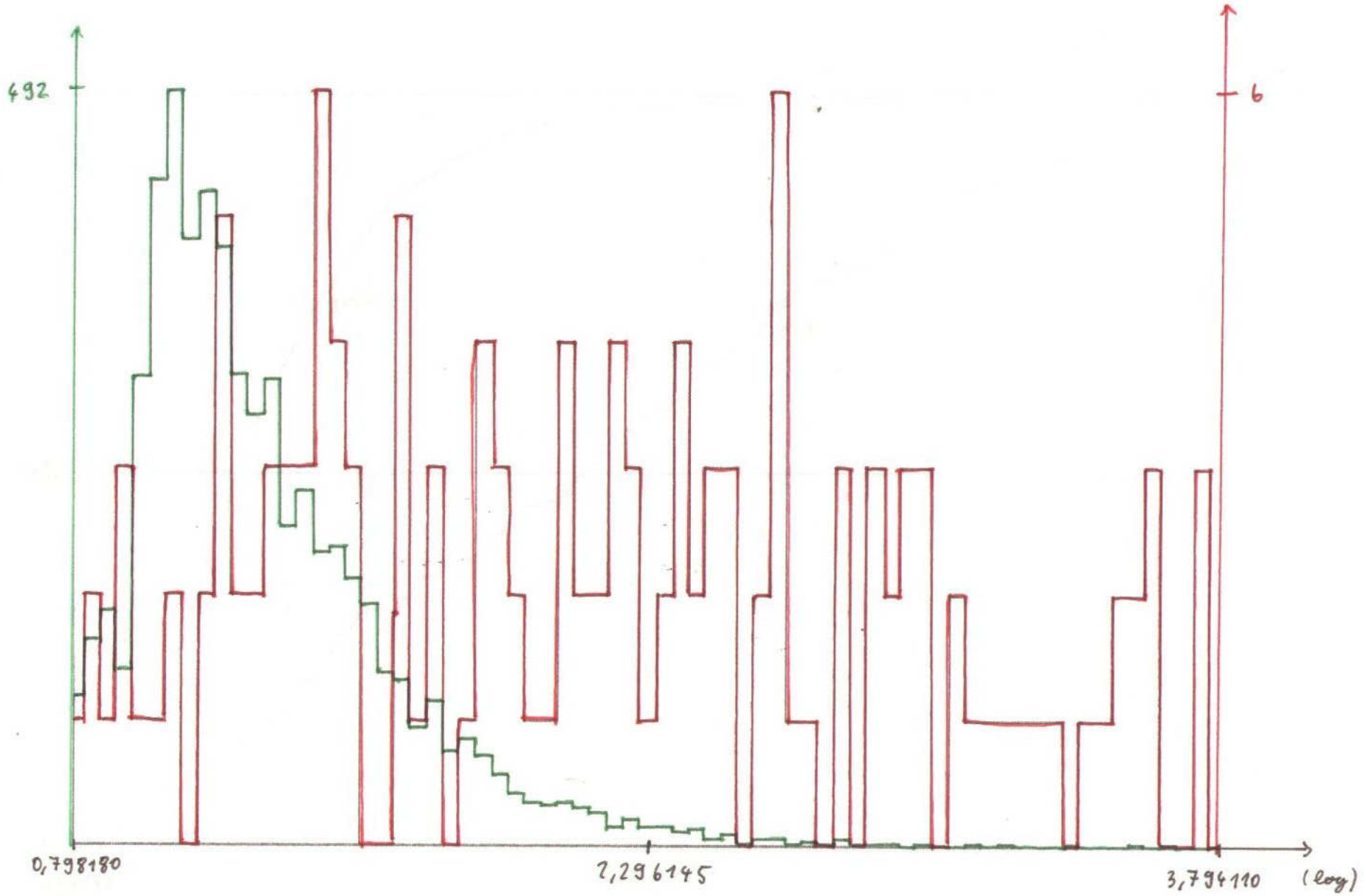
Anfangswert: 2.400000 Endwert: 2.500000
Intervallbreite: 0.005000

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
62.59%	37.41%	98.97%	1.03%	1.47%	46.02%	2.400000
62.59%	37.41%	99.03%	0.97%	1.47%	47.71%	2.405000
63.31%	36.69%	99.05%	0.95%	1.48%	47.66%	2.410000
64.03%	35.97%	99.07%	0.93%	1.50%	47.62%	2.415000
64.03%	35.97%	99.08%	0.92%	1.50%	48.08%	2.420000
64.03%	35.97%	99.10%	0.90%	1.50%	48.54%	2.425000
64.03%	35.97%	99.10%	0.90%	1.50%	48.54%	2.430000
64.03%	35.97%	99.12%	0.88%	1.50%	49.02%	2.435000
64.03%	35.97%	99.15%	0.85%	1.50%	50.00%	2.440000
64.03%	35.97%	99.17%	0.83%	1.50%	50.51%	2.445000
64.03%	35.97%	99.20%	0.80%	1.50%	51.55%	2.450000
64.03%	35.97%	99.20%	0.80%	1.50%	51.55%	2.455000
64.03%	35.97%	99.20%	0.80%	1.50%	51.55%	2.460000
64.75%	35.25%	99.20%	0.80%	1.51%	51.04%	2.465000

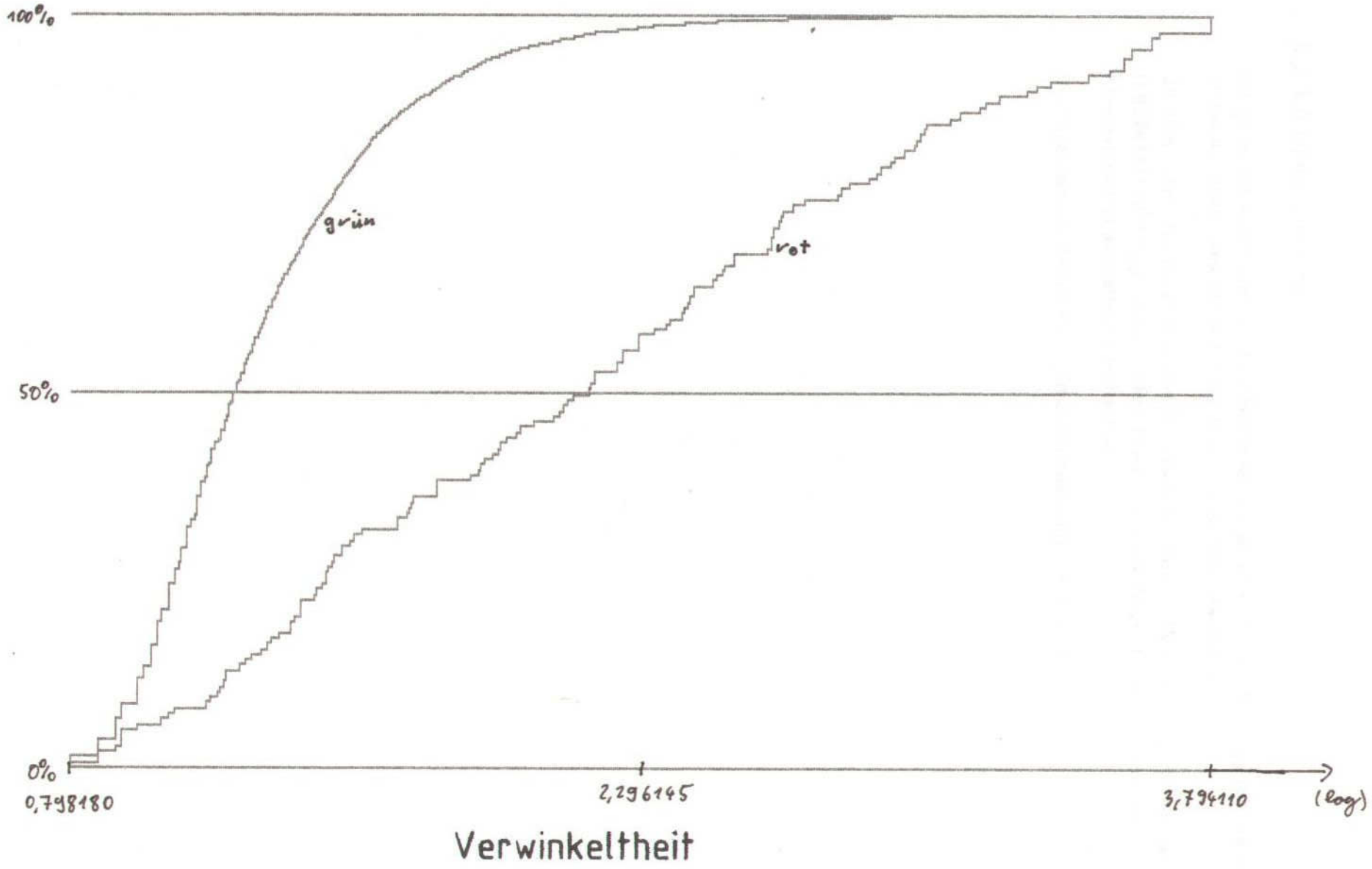
64.75%	35.25%	99.22%	0.78%	1.51%	51.58%	2.470000
65.47%	34.53%	99.22%	0.78%	1.53%	51.06%	2.475000
65.47%	34.53%	99.22%	0.78%	1.53%	51.06%	2.480000
66.19%	33.81%	99.24%	0.76%	1.55%	51.09%	2.485000
66.19%	33.81%	99.27%	0.73%	1.55%	52.22%	2.490000
66.91%	33.09%	99.27%	0.73%	1.56%	51.69%	2.495000
66.91%	33.09%	99.29%	0.71%	1.56%	52.27%	2.500000

Anfangswert: 0.798180 Endwert: 1.050000
Intervallbreite: 0.012591

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
0.72%	99.28%	1.64%	98.36%	1.02%	2.32%	0.798180
0.72%	99.28%	1.64%	98.36%	1.02%	2.32%	0.810771
0.72%	99.28%	1.64%	98.36%	1.02%	2.32%	0.823362
0.72%	99.28%	1.64%	98.36%	1.02%	2.32%	0.835953
0.72%	99.28%	1.64%	98.36%	1.02%	2.32%	0.848544
0.72%	99.28%	1.64%	98.36%	1.02%	2.32%	0.861135
2.16%	97.84%	3.92%	96.08%	1.28%	2.34%	0.873726
2.16%	97.84%	3.92%	96.08%	1.28%	2.34%	0.886317
2.16%	97.84%	3.92%	96.08%	1.28%	2.34%	0.898908
2.16%	97.84%	3.92%	96.08%	1.28%	2.34%	0.911499
2.88%	97.12%	6.49%	93.51%	1.03%	2.39%	0.924090
5.04%	94.96%	8.42%	91.58%	1.39%	2.38%	0.936681
5.04%	94.96%	8.42%	91.58%	1.39%	2.38%	0.949272
5.04%	94.96%	8.42%	91.58%	1.39%	2.38%	0.961863
5.76%	94.24%	12.07%	87.93%	1.11%	2.46%	0.974454
5.76%	94.24%	12.07%	87.93%	1.11%	2.46%	0.987045
5.76%	94.24%	13.59%	86.41%	0.99%	2.51%	0.999636
5.76%	94.24%	16.27%	83.73%	0.83%	2.58%	1.012227
5.76%	94.24%	19.49%	80.51%	0.69%	2.68%	1.024818
6.47%	93.53%	20.95%	79.05%	0.72%	2.71%	1.037409
6.47%	93.53%	20.95%	79.05%	0.72%	2.71%	1.050000



Verwinkeltheit


100%
50%
0%
grün
rot
0,798180
2,296145
3,794110
(log)
Verwinkeltheit

6.3.2.4.Umfang/Fläche

Regionen mit einem Wert Umfang/Fläche kleiner gleich 0.2995($*10^{5*1/m}$) sind zu 50% Extravasationen. Dies ist der Fall für 41.73% aller Extravasationen.

26.88% der Nicht-Extravasationen besitzen einen Wert Umfang/Fläche größer 0.932414($*10^{5*1/m}$). 2.88% aller Extravasationen fallen in diesen Bereich. Dieses Kriterium ist hier sehr effektiv anwendbar.

Die folgenden vier Seiten enthalten die Auswertung dieser Größe.

Datenbezeichnung: Umfang/Flaeche; Anzahl der Daten 1: 139
Datenbezeichnung: Umfang/Flaeche; Anzahl der Daten 2: 5900
70 Säulen Lineare Darstellung
Reales Minimum: 0.095654 Reales Maximum: 1.329655
Minimum im Diagramm: 0.095654 Maximum im Diagramm: 1.347284
Intervallbreite: 0.017629
Histogramm 1 (rot) : Maximum = 10
Histogramm 2 (gruen) : Maximum = 231
Angeglichene Histogramme

Anfangswert: 0.095654 Endwert: 1.329655
Intervallbreite: 0.061700

'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
0.72%	99.28%	0.00%	100.00%	100.00%	2.29%	0.095654
7.19%	92.81%	0.03%	99.97%	83.33%	2.14%	0.157354
20.14%	79.86%	0.22%	99.78%	68.29%	1.85%	0.219054
36.69%	63.31%	0.75%	99.25%	53.68%	1.48%	0.280754
46.04%	53.96%	2.15%	97.85%	33.51%	1.28%	0.342454
57.55%	42.45%	4.58%	95.42%	22.86%	1.04%	0.404154
61.87%	38.13%	8.76%	91.24%	14.26%	0.97%	0.465854
69.78%	30.22%	14.08%	85.92%	10.45%	0.82%	0.527554
79.86%	20.14%	20.66%	79.34%	8.35%	0.59%	0.589254
82.01%	17.99%	28.53%	71.47%	6.34%	0.59%	0.650954
89.21%	10.79%	36.59%	63.41%	5.43%	0.40%	0.712654
92.09%	7.91%	45.36%	54.64%	4.56%	0.34%	0.774355
94.96%	5.04%	55.64%	44.36%	3.87%	0.27%	0.836055
95.68%	4.32%	67.05%	32.95%	3.25%	0.31%	0.897755
98.56%	1.44%	77.32%	22.68%	2.92%	0.15%	0.959455
99.28%	0.72%	86.92%	13.08%	2.62%	0.13%	1.021155
99.28%	0.72%	93.71%	6.29%	2.44%	0.27%	1.082855
100.00%	0.00%	97.39%	2.61%	2.36%	0.00%	1.144555
100.00%	0.00%	99.15%	0.85%	2.32%	0.00%	1.206255
100.00%	0.00%	99.83%	0.17%	2.31%	0.00%	1.267955
100.00%	0.00%	100.00%	0.00%	2.30%	999.00%	1.329655

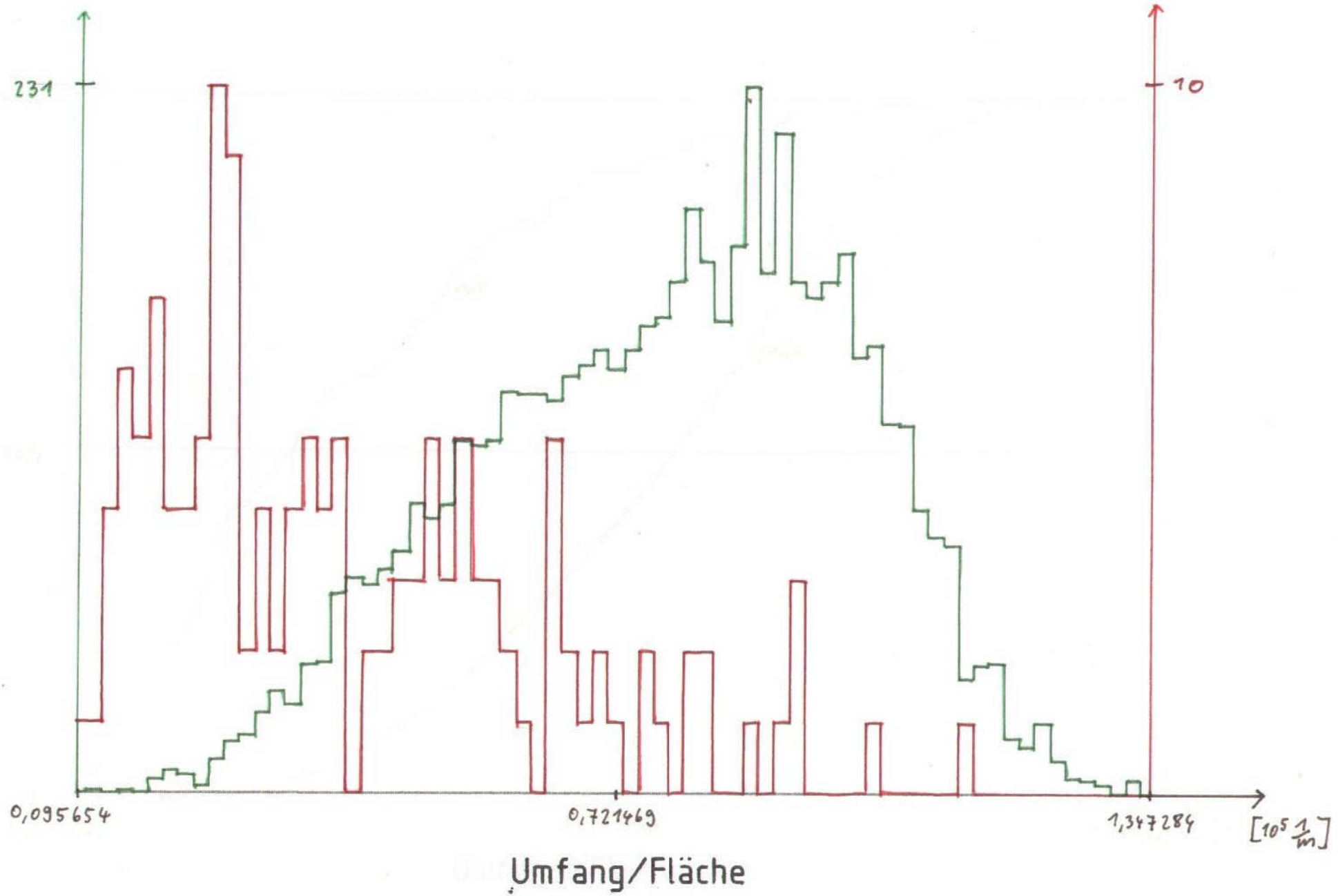
Anfangswert: 0.280000 Endwert: 0.310000
Intervallbreite: 0.001500

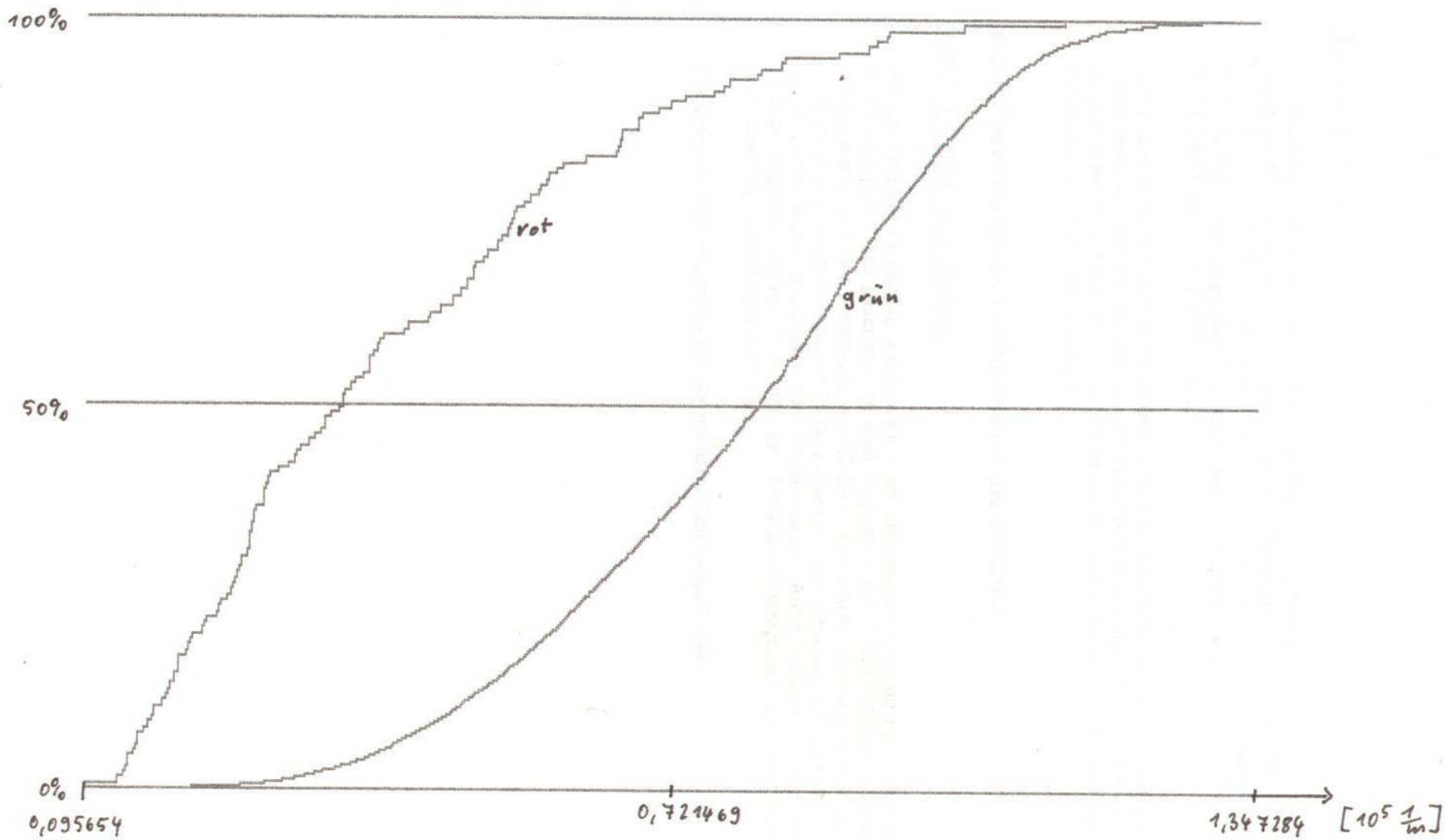
'1'<=	'1'>	'2'<=	'2'>	'p1'<=	'p1'>	als
36.69%	63.31%	0.75%	99.25%	53.68%	1.48%	0.280000
36.69%	63.31%	0.75%	99.25%	53.68%	1.48%	0.281500
36.69%	63.31%	0.75%	99.25%	53.68%	1.48%	0.283000
38.85%	61.15%	0.76%	99.24%	54.55%	1.43%	0.284500
38.85%	61.15%	0.78%	99.22%	54.00%	1.43%	0.286000
39.57%	60.43%	0.80%	99.20%	53.92%	1.41%	0.287500
40.29%	59.71%	0.83%	99.17%	53.33%	1.40%	0.289000
41.01%	58.99%	0.83%	99.17%	53.77%	1.38%	0.290500
41.01%	58.99%	0.85%	99.15%	53.27%	1.38%	0.292000
41.01%	58.99%	0.86%	99.14%	52.78%	1.38%	0.293500
41.01%	58.99%	0.90%	99.10%	51.82%	1.38%	0.295000
41.01%	58.99%	0.95%	99.05%	50.44%	1.38%	0.296500
41.73%	58.27%	0.98%	99.02%	50.00%	1.37%	0.298000
41.73%	58.27%	0.98%	99.02%	50.00%	1.37%	0.299500

41.73%	58.27%	1.03%	98.97%	48.74%	1.37%	0.301000
41.73%	58.27%	1.07%	98.93%	47.93%	1.37%	0.302500
41.73%	58.27%	1.08%	98.92%	47.54%	1.37%	0.304000
41.73%	58.27%	1.12%	98.88%	46.77%	1.37%	0.305500
41.73%	58.27%	1.15%	98.85%	46.03%	1.37%	0.307000
41.73%	58.27%	1.19%	98.81%	45.31%	1.37%	0.308500
42.45%	57.55%	1.22%	98.78%	45.04%	1.35%	0.310000

Anfangswert: 0.800000 Endwert: 1.329655
Intervallbreite: 0.026483

'i'<=	'i'>	'2'<=	'2'>	'p1'<=	'p1'>	als
92.09%	7.91%	49.32%	50.68%	4.21%	0.37%	0.800000
94.24%	5.76%	53.31%	46.69%	4.00%	0.29%	0.826483
94.96%	5.04%	58.12%	41.88%	3.71%	0.28%	0.852966
94.96%	5.04%	62.85%	37.15%	3.44%	0.32%	0.879448
95.68%	4.32%	68.03%	31.97%	3.21%	0.32%	0.905931
97.12%	2.88%	73.12%	26.88%	3.03%	0.25%	0.932414
98.56%	1.44%	77.24%	22.76%	2.92%	0.15%	0.958897
98.56%	1.44%	81.51%	18.49%	2.77%	0.18%	0.985379
98.56%	1.44%	85.58%	14.42%	2.64%	0.23%	1.011862
99.28%	0.72%	88.90%	11.10%	2.56%	0.15%	1.038345
99.28%	0.72%	92.10%	7.90%	2.48%	0.21%	1.064828
99.28%	0.72%	94.39%	5.61%	2.42%	0.30%	1.091310
99.28%	0.72%	96.51%	3.49%	2.37%	0.48%	1.117793
100.00%	0.00%	97.39%	2.61%	2.36%	0.00%	1.144276
100.00%	0.00%	98.59%	1.41%	2.33%	0.00%	1.170759
100.00%	0.00%	99.07%	0.93%	2.32%	0.00%	1.197241
100.00%	0.00%	99.54%	0.46%	2.31%	0.00%	1.223724
100.00%	0.00%	99.76%	0.24%	2.31%	0.00%	1.250207
100.00%	0.00%	99.88%	0.12%	2.30%	0.00%	1.276690
100.00%	0.00%	99.93%	0.07%	2.30%	0.00%	1.303172
100.00%	0.00%	100.00%	0.00%	2.30%	999.00%	1.329655


231
10
0,095654
0,721469
1,347284
[10⁵ 1/m]
Umfang/Fläche


100%
50%
0%
rot
grün
0,095654
0,721469
1,347284
[10⁵ 1/m]
Umfang/Fläche

6.3.2.5.Ergebnis

Als Ergebnis muß man sagen, daß sich die Fläche als geeignetstes Kriterium zur Entscheidung, ob es sich um eine Extravasation handelt, herauskristallisiert hat. So kann man z.B. bei 18.71% aller Extravasationen sagen, daß sie mit einer Wahrscheinlichkeit von 81.25% Extravasationen sind. Die Fläche beträgt dabei mehr als $10^{3.74596}$(*100µm²)= 5571.34(*100µm²) = 0.557134mm².

Durch den Wert Umfang/Fläche lassen sich am besten Nicht-Extravasationen eliminieren, wenn man in Kauf nimmt, daß einige Extravasationen dabei miteliminiert werden. Bei obigen Werten werden 1586 Nicht-Extravasationen erkannt. Zusätzlich werden vier Extravasationen fälschlich erkannt.

6.3.3.Auswertung von Kombinationen der Kriterien

6.3.3.1.ODER-Verknüpfung

Bei der ODER-Verknüpfung werden die oben gefundenen Kriterien mit einem logischen ODER verknüpft. Dies geschieht mit dem Programm ZUS_ODER.C (siehe Kapitel 7.13.). Hierbei ergibt sich eine Erkennung von 50.36% aller Extravasationen, allerdings sind nur 37.63% aller erkannten Regionen Extravasationen. D.h. diese Art der Verknüpfung ergibt keine Verbesserung gegenüber den Einzelkriterien. Auch werden bei der Erkennung anderer Regionen zusätzlich 8.63% der Extravasationsregionen erkannt. Dieses stellt doch einen zu hohen Prozentsatz dar.

Die gesamte Auswertung ist auf der nächsten Seite abgedruckt.

ODER-Verknüpfung

Extravasationsregionenerkennung:
Untere Grenze für die Fläche (log) : 3.150000
Untere Grenze für das Maximum (log) : 1.925000
Untere Grenze für den Umfang (log) : 2.800000
Untere Grenze für die Verwinkeltheit (log) : 2.440000
Obere Grenze für Umfang/Fläche : 0.299500

Erkennung anderer Regionen:
Obere Grenze für die Fläche (log) : 1.486895
Obere Grenze für den Mittelwert : 12.613793
Obere Grenze für das Maximum (log) : 1.220618
Obere Grenze für den Umfang (log) : 1.430855
Untere Grenze für die Rundheit (log) : 0.023674
Obere Grenze für die Verwinkeltheit (log) : 0.911499
Untere Grenze für Umfang/Fläche : 0.932414

Anzahl der Extravasationsregionen: 139
Anzahl der anderen Regionen : 5900

Erkannter Prozentsatz der Extravasationsregionen : 50.36%
Dabei falsch erkannter Prozentsatz anderer Regionen : 1.97%
Prozentsatz der Extravasationen von den Regionen,
die die Bedingung erfüllen : 37.63%

Erkannter Prozentsatz der anderen Regionen : 43.73%
Dabei falsch erkannter Prozentsatz der Extravasationsregionen: 8.63%
Prozentsatz der anderen Regionen von den Regionen,
die die Bedingung erfüllen : 99.54%

6.3.3.2.UND-Verknüpfung

Bei der UND-Verknüpfung werden die oben gefundenen Kriterien mit einem logischen UND verknüpft. Dies geschieht mit dem Programm ZUS_UND.C (siehe Kapitel 7.14.). Dabei werden jedoch zu wenige Regionen erkannt, so daß diese Art der Auswertung nicht geeignet ist.

Die gesamte Auswertung ist auf der nächsten Seite abgedruckt.

UND-Verknüpfung

Extravasationsregionenerkennung:
Untere Grenze für die Fläche (log) : 3.150000
Untere Grenze für das Maximum (log) : 1.925000
Untere Grenze für den Umfang (log) : 2.800000
Untere Grenze für die Verwinkeltheit (log) : 2.440000
Obere Grenze für Umfang/Fläche : 0.299500

Erkennung anderer Regionen:
Obere Grenze für die Fläche (log) : 1.486895
Obere Grenze für den Mittelwert : 12.613793
Obere Grenze für das Maximum (log) : 1.220618
Obere Grenze für den Umfang (log) : 1.430855
Untere Grenze für die Rundheit (log) : 0.023674
Obere Grenze für die Verwinkeltheit (log) : 0.911499
Untere Grenze für Umfang/Fläche : 0.932414

Anzahl der Extravasationsregionen: 139
Anzahl der anderen Regionen : 5900

Erkannter Prozentsatz der Extravasationsregionen : 2.88%
Dabei falsch erkannter Prozentsatz anderer Regionen : 0.02%
Prozentsatz der Extravasationen von den Regionen,
die die Bedingung erfüllen : 80.00%

Erkannter Prozentsatz der anderen Regionen : 0.07%
Dabei falsch erkannter Prozentsatz der Extravasationsregionen: 0.00%
Prozentsatz der anderen Regionen von den Regionen,
die die Bedingung erfüllen : 100.00%

6.3.3.3.Ketten-Verknüpfung

Bei der Ketten-Verknüpfung werden zwei der oben gefundenen Kriterien nacheinander angewendet, um eine eventuelle Verbesserung nachvollziehen zu können. Dies geschieht mit dem Programm ZUS_KET.C (siehe Kapitel 7.15.). Um eine Verbesserung der Erkennungsrate von Extravasationen zu erzielen, ist es notwendig, daß der Prozentsatz der zusätzlich erkannten Extravasationsregionen von allen zusätzlich (d.h. vom Sekundärkriterium) erkannten Regionen mehr als 50% beträgt. Dies ist aber bei keiner Verknüpfung der Fall.
Auch bei der Erkennung anderer Regionen läßt sich keine Verbesserung erzielen, da dider Prozentsatz der zusätzlich (fälschlich) erkannten Extravasationsregionen unverhältnismäßig stark ansteigt.

Auf den folgenden sieben Seiten sind ausgewählte Ketten-Verknüpfungen protokolliert, die hier aber nur Beispielcharacter besitzen sollen.

Ketten-Verknüpfung

Extravasationsregionenerkennung:
Untere Grenze für die Fläche (log) : 3.150000
Obere Grenze für Umfang/Fläche : 0.299500

Erkennung anderer Regionen:
Obere Grenze für die Fläche (log) : 1.486895
Untere Grenze für Umfang/Fläche : 0.932414

Anzahl der Extravasationsregionen: 139
Anzahl der anderen Regionen : 5900
Primärdaten : Flaeche
Sekundärdaten : Umfang/Flaeche

Erkannter Prozentsatz der Extravasationsregionen
mit Primärdaten: 45.32%
Zusätzlich erkannte Extravasationsregionen mit Sekundärdaten: 3.60%
Bereits erkannte Extravasationsregionen mit Sekundärdaten: 38.13%
Dabei falsch erkannte andere Regionen mit Primärdaten: 1.07%
Zusätzlich falsch erkannte andere Regionen mit Sekundärdaten: 0.61%
Bereits falsch erkannte andere Regionen mit Sekundärdaten: 0.37%

Prozentsatz der Extravasationen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 50.00%
Prozentsatz der Extravasationen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 12.20%
Prozentsatz der Extravasationen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 70.67%

Erkannter Prozentsatz der anderen Regionen mit Primärdaten: 14.32%
Zusätzlich erkannte andere Regionen mit Sekundärdaten: 14.95%
Bereits erkannte andere Regionen mit Sekundärdaten: 11.93%
Dabei falsch erkannte Extravasationsregionen mit Primärdaten: 2.16%
Zusätzlich falsch erkannte Extravasationsregionen
mit Sekundärdaten: 2.16%
Bereits falsch erkannte Extravasationsregionen mit
Sekundärdaten: 0.72%

Prozentsatz der anderen Regionen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 99.65%
Prozentsatz der anderen Regionen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 99.66%
Prozentsatz der anderen Regionen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 99.86%

Ketten-Verknüpfung
==========

Extravasationsregionenerkennung:
Untere Grenze für die Fläche (log) : 3.150000
Obere Grenze für Umfang/Fläche : 0.299500

Erkennung anderer Regionen:
Obere Grenze für die Fläche (log) : 1.486895
Untere Grenze für Umfang/Fläche : 0.932414

Anzahl der Extravasationsregionen: 139
Anzahl der anderen Regionen : 5900
Primärdaten : Umfang/Flaeche
Sekundärdaten : Flaeche

Erkannter Prozentsatz der Extravasationsregionen
mit Primärdaten: 41.73%
Zusätzlich erkannte Extravasationsregionen mit Sekundärdaten: 7.19%
Bereits erkannte Extravasationsregionen mit Sekundärdaten: 38.13%
Dabei falsch erkannte andere Regionen mit Primärdaten: 0.98%
Zusätzlich falsch erkannte andere Regionen mit Sekundärdaten: 0.69%
Bereits falsch erkannte andere Regionen mit Sekundärdaten: 0.37%

Prozentsatz der Extravasationen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 50.00%
Prozentsatz der Extravasationen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 19.61%
Prozentsatz der Extravasationen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 70.67%

Erkannter Prozentsatz der anderen Regionen mit Primärdaten: 26.88%
Zusätzlich erkannte andere Regionen mit Sekundärdaten: 2.39%
Bereits erkannte andere Regionen mit Sekundärdaten: 11.93%
Dabei falsch erkannte Extravasationsregionen mit Primärdaten: 2.88%
Zusätzlich falsch erkannte Extravasationsregionen
mit Sekundärdaten: 1.44%
Bereits falsch erkannte Extravasationsregionen mit
Sekundärdaten: 0.72%

Prozentsatz der anderen Regionen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 99.75%
Prozentsatz der anderen Regionen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 98.60%
Prozentsatz der anderen Regionen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 99.86%

Ketten-Verknüpfung

Extravasationsregionenerkennung:
Untere Grenze für den Umfang (log) : 2.800000
Obere Grenze für Umfang/Fläche : 0.299500

Erkennung anderer Regionen:
Obere Grenze für den Umfang (log) : 1.430855
Untere Grenze für Umfang/Fläche : 0.932414

Anzahl der Extravasationsregionen: 139
Anzahl der anderen Regionen : 5900
Primärdaten : Umfang/Flaeche
Sekundärdaten : Umfang

Erkannter Prozentsatz der Extravasationsregionen
mit Primärdaten: 41.73%
Zusätzlich erkannte Extravasationsregionen mit Sekundärdaten: 5.76%
Bereits erkannte Extravasationsregionen mit Sekundärdaten: 27.34%
Dabei falsch erkannte andere Regionen mit Primärdaten: 0.98%
Zusätzlich falsch erkannte andere Regionen mit Sekundärdaten: 0.61%
Bereits falsch erkannte andere Regionen mit Sekundärdaten: 0.17%

Prozentsatz der Extravasationen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 50.00%
Prozentsatz der Extravasationen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 18.18%
Prozentsatz der Extravasationen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 79.17%

Erkannter Prozentsatz der anderen Regionen mit Primärdaten: 26.88%
Zusätzlich erkannte andere Regionen mit Sekundärdaten: 2.78%
Bereits erkannte andere Regionen mit Sekundärdaten: 2.61%
Dabei falsch erkannte Extravasationsregionen mit Primärdaten: 2.88%
Zusätzlich falsch erkannte Extravasationsregionen
mit Sekundärdaten: 1.44%
Bereits falsch erkannte Extravasationsregionen mit
Sekundärdaten: 0.72%

Prozentsatz der anderen Regionen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 99.75%
Prozentsatz der anderen Regionen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 98.80%
Prozentsatz der anderen Regionen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 99.35%

Ketten-Verknüpfung

Extravasationsregionenerkennung:
Untere Grenze für die Verwinkeltheit (log) : 2.440000
Obere Grenze für Umfang/Fläche : 0.299500

Erkennung anderer Regionen:
Obere Grenze für die Verwinkeltheit (log) : 0.911499
Untere Grenze für Umfang/Fläche : 0.932414

Anzahl der Extravasationsregionen: 139
Anzahl der anderen Regionen : 5900
Primärdaten : Umfang/Flaeche
Sekundärdaten : Verwinkeltheit

Erkannter Prozentsatz der Extravasationsregionen
mit Primärdaten: 41.73%
Zusätzlich erkannte Extravasationsregionen mit Sekundärdaten: 7.19%
Bereits erkannte Extravasationsregionen mit Sekundärdaten: 28.78%
Dabei falsch erkannte andere Regionen mit Primärdaten: 0.98%
Zusätzlich falsch erkannte andere Regionen mit Sekundärdaten: 0.66%
Bereits falsch erkannte andere Regionen mit Sekundärdaten: 0.19%

Prozentsatz der Extravasationen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 50.00%
Prozentsatz der Extravasationen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 20.41%
Prozentsatz der Extravasationen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 79.43%

Erkannter Prozentsatz der anderen Regionen mit Primärdaten: 26.88%
Zusätzlich erkannte andere Regionen mit Sekundärdaten: 2.49%
Bereits erkannte andere Regionen mit Sekundärdaten: 1.42%
Dabei falsch erkannte Extravasationsregionen mit Primärdaten: 2.88%
Zusätzlich falsch erkannte Extravasationsregionen
mit Sekundärdaten: 1.44%
Bereits falsch erkannte Extravasationsregionen mit
Sekundärdaten: 0.72%

Prozentsatz der anderen Regionen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 99.75%
Prozentsatz der anderen Regionen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 98.66%
Prozentsatz der anderen Regionen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 98.82%

Ketten-Verknüpfung

Extravasationsregionenerkennung:
Untere Grenze für die Fläche (log) : 3.150000
Untere Grenze für die Verwinkeltheit (log) : 2.440000

Erkennung anderer Regionen:
Obere Grenze für die Fläche (log) : 1.486895
Obere Grenze für die Verwinkeltheit (log) : 0.911499

Anzahl der Extravasationsregionen: 139
Anzahl der anderen Regionen : 5900
Primärdaten : Flaeche
Sekundärdaten : Verwinkeltheit

Erkannter Prozentsatz der Extravasationsregionen
mit Primärdaten: 45.32%
Zusätzlich erkannte Extravasationsregionen mit Sekundärdaten: 1.44%
Bereits erkannte Extravasationsregionen mit Sekundärdaten: 34.53%
Dabei falsch erkannte andere Regionen mit Primärdaten: 1.07%
Zusätzlich falsch erkannte andere Regionen mit Sekundärdaten: 0.20%
Bereits falsch erkannte andere Regionen mit Sekundärdaten: 0.64%

Prozentsatz der Extravasationen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 50.00%
Prozentsatz der Extravasationen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 14.29%
Prozentsatz der Extravasationen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 55.81%

Erkannter Prozentsatz der anderen Regionen mit Primärdaten: 14.32%
Zusätzlich erkannte andere Regionen mit Sekundärdaten: 1.49%
Bereits erkannte andere Regionen mit Sekundärdaten: 2.42%
Dabei falsch erkannte Extravasationsregionen mit Primärdaten: 2.16%
Zusätzlich falsch erkannte Extravasationsregionen
mit Sekundärdaten: 2.16%
Bereits falsch erkannte Extravasationsregionen mit
Sekundärdaten: 0.00%

Prozentsatz der anderen Regionen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 99.65%
Prozentsatz der anderen Regionen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 96.70%
Prozentsatz der anderen Regionen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 100.00%

Ketten-Verknüpfung

Extravasationsregionenerkennung:
Untere Grenze für den Umfang (log) : 2.800000
Untere Grenze für die Verwinkeltheit (log) : 2.440000

Erkennung anderer Regionen:
Obere Grenze für den Umfang (log) : 1.430855
Obere Grenze für die Verwinkeltheit (log) : 0.911499

Anzahl der Extravasationsregionen: 139
Anzahl der anderen Regionen : 5900
Primärdaten : Umfang
Sekundärdaten : Verwinkeltheit

Erkannter Prozentsatz der Extravasationsregionen
mit Primärdaten: 33.09%
Zusätzlich erkannte Extravasationsregionen mit Sekundärdaten: 4.32%
Bereits erkannte Extravasationsregionen mit Sekundärdaten: 31.65%
Dabei falsch erkannte andere Regionen mit Primärdaten: 0.78%
Zusätzlich falsch erkannte andere Regionen mit Sekundärdaten: 0.22%
Bereits falsch erkannte andere Regionen mit Sekundärdaten: 0.63%

Prozentsatz der Extravasationen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 50.00%
Prozentsatz der Extravasationen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 31.58%
Prozentsatz der Extravasationen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 54.32%

Erkannter Prozentsatz der anderen Regionen mit Primärdaten: 5.39%
Zusätzlich erkannte andere Regionen mit Sekundärdaten: 1.98%
Bereits erkannte andere Regionen mit Sekundärdaten: 1.93%
Dabei falsch erkannte Extravasationsregionen mit Primärdaten: 2.16%
Zusätzlich falsch erkannte Extravasationsregionen
mit Sekundärdaten: 2.16%
Bereits falsch erkannte Extravasationsregionen mit
Sekundärdaten: 0.00%

Prozentsatz der anderen Regionen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 99.07%
Prozentsatz der anderen Regionen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 97.50%
Prozentsatz der anderen Regionen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 100.00%

Ketten-Verknüpfung
==========

Extravasationsregionenerkennung:
Untere Grenze für den Umfang (log) : 2.800000
Untere Grenze für die Verwinkeltheit (log) : 2.440000

Erkennung anderer Regionen:
Obere Grenze für den Umfang (log) : 1.430855
Obere Grenze für die Verwinkeltheit (log) : 0.911499

Anzahl der Extravasationsregionen: 139
Anzahl der anderen Regionen : 5900
Primärdaten : Verwinkeltheit
Sekundärdaten : Umfang

Erkannter Prozentsatz der Extravasationsregionen
mit Primärdaten: 35.97%
Zusätzlich erkannte Extravasationsregionen mit Sekundärdaten: 1.44%
Bereits erkannte Extravasationsregionen mit Sekundärdaten: 31.65%
Dabei falsch erkannte andere Regionen mit Primärdaten: 0.85%
Zusätzlich falsch erkannte andere Regionen mit Sekundärdaten: 0.15%
Bereits falsch erkannte andere Regionen mit Sekundärdaten: 0.63%

Prozentsatz der Extravasationen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 50.00%
Prozentsatz der Extravasationen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 18.18%
Prozentsatz der Extravasationen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 54.32%

Erkannter Prozentsatz der anderen Regionen mit Primärdaten: 3.92%
Zusätzlich erkannte andere Regionen mit Sekundärdaten: 3.46%
Bereits erkannte andere Regionen mit Sekundärdaten: 1.93%
Dabei falsch erkannte Extravasationsregionen mit Primärdaten: 2.16%
Zusätzlich falsch erkannte Extravasationsregionen
mit Sekundärdaten: 2.16%
Bereits falsch erkannte Extravasationsregionen mit
Sekundärdaten: 0.00%

Prozentsatz der anderen Regionen von allen Regionen,
die die Bedingung der Primärdaten erfüllen: 98.72%
Prozentsatz der anderen Regionen von allen Regionen,
die zusätzlich die Bedingung der Sekundärdaten erfüllen: 98.55%
Prozentsatz der anderen Regionen von allen Regionen,
die bereits die Bedingung der Sekundärdaten erfüllten: 100.00%

6.3.4.Zusammenfassung

Nach diesen Untersuchungen läßt sich sagen, daß eine gute Klassifikation mit den verwendeten Kriterien nicht möglich ist. Überraschend ist die Erkenntnis, daß sich Extravasationsregionen nicht durch die Länglichkeit, bzw. die Rundheit von anderen Regionen unterscheiden lassen, da der Ansatz zu diesen Untersuchungen war, daß sich Extravasationen spotförmig, also rund äußern, und Dilatationen, die den Großteil der anderen Regionen ausmachen, eher länglichen Charakter besitzen. Dies konnte wohl dadurch nicht erkannt werden, da sich Extravasationen entlang Gefäßkanten ausbilden und daher aus länglichen Regionen heraus wachsen.
Für die praktische Verwertung dieser Ergebnisse lassen sich wohl nur die über die Erkennung von Nicht-Extravasationsregionen ausnützen. Da diese Störfaktoren für die Versuche darstellen, will man sie reduzieren. Dies ist zum Beispiel über das Kriterium Umfang/Fläche möglich, wie in Kapitel 6.3.1.9. beschrieben. Dabei muß man aber in Kauf nehmen, interessierende Regionen, wenn auch in geringem Maße, zu löschen.
Über die direkte Erkennung von Extravasationsregionen läuft man Gefahr, nicht einmal die Hälfte aller interessierenden Regionen zu erkennen und gleichzeitig genügend viele Nicht-Extravasationen im Bild zu hinterlassen, so daß das Ergebnis zu stark verfälscht wäre.
Die Auswertung hat außerdem gezeigt, daß die beobachtete Hirnoberfläche vor jeder Aufnahme penibelst genau gereinigt werden muß. Dies ist notwendig, um das Bild von Verunreinigungen zu befreien, die eine Extravasation vortäuschen können.
Weiterhin sollte bei kommenden Versuchen die Zeiteinblendung aus dem Bild verbannt werden und durch Synchronbesprechung des Videobandes ersetzt werden. Außerdem sollte man versuchen, mehrere Bilder zu mitteln (also mehr als 4), um ein rauschärmeres und homogeneres Bild zu erhalten.

7.Realisierungen

Zum Verständnis der Programme sind die folgenden allgemeinen Zusatzinformationen notwendig:

```
#define MAXPAR 12     /* max. Anzahl der Parameter für die Lagekorrektur      */
                      /* in estpar und imtran                                 */
long Ze=512L;  /* Anzahl der Zeilen der Bilder                                */
long Sp=503L;  /* Anzahl der Spalten der Bilder                    */
long La=257536L;      /* =Ze*Sp Anzahl der Bildpixel                         */
char Datum[6];        /* Datum des Versuchs                                  */
char Dpfad[20]={"C:\\DATEN\\"};       /* Datenpfad                           */
char Bpfad[20]={"D:\\PIC\\"};         /* Bildpfad                            */
double pixelbreite=1.0/0.82;          /* Pixelbreite in 1/100 mm             */
double pixelhoehe=1.0/1.265;   /* Pixelhoehe in 1/100 mm                     */
struct ffblk ffblk;/* Zugriff auf aktuellen Filenamen ffblk.ff_name[13]      */

/*************************************************************************/

void write_Frame(int X, int Y, int Value);
/* Schreibt den angegebenen Wert an die angegebene Stelle des Bild-   */
/* kartenspeichers                                                    */
/* X              :Spalte                                             */
/* Y              :Zeile                                              */
/* Value:Wert                                                         */

int read_Frame(int X, int Y);
/* Funktion, die den Wert an der angegebenen Stelle im Bildkartenspeicher */
/* ausgibt                                                            */
/* X              :Spalte                                             */
/* Y              :Zeile                                              */
```

7.1.Shadingkorrektur

Diese Funktion bewirkt das Einlesen des benötigten Korrekturbildes von der Festplatte in den Bildkartenspeicher und das Shadingkorrigieren des im RAM befindlichen Bildes. Um das Einlesen so schnell wie möglich zu gestalten, wird der gesamte freie Speicherraum im addressierbaren Speicher ausgenutzt (Befehl: farcoreleft()). Danach wird das Maximum bestimmt. Da das Korrekturbild sein Maximum im Bereich der Bildmitte besitzen muß, wird die Suche danach auf das innere Neuntel des in neun gleiche Teile geteilten Bildes beschränkt. Damit wird dann mit oben genannter Formel eine Shadingkorrektur durchgeführt. Falls, z.B. durch eine Störung im Bild, das Korrekturbild den Grauwert Null besitzen sollte, so wird diese Stelle im korrigierten Bild auf Null gesetzt, damit sie keinen weiteren Einfluß nehmen kann.

```
void bild_einzel_shading_korrigieren(unsigned char huge *Bild,int sb,int zb);
/* Shading-korrigieren eines Bildes; mediangefilterte Topfbilder sind
        */
/* unter 'Datum'?1.MED abgespeichert                                        */
/* Das Topfbild wird in den Bildspeicher geladen                            */
/* Bild         :zu korrigierendes Bild
        */
/* sb           :Spaltenoffset des in den Bildspeicher einzulesenden Topfbildes  */
/* zb           :Zeilenoffset des  in den Bildspeicher einzulesenden Topfbildes  */

/*************************************************************************/

void bild_einzel_shading_korrigieren(unsigned char huge *Bild,int sb,int zb)
{
int laenge,si,zi;
unsigned char max=0;
char bildname[30];
int Dateibin;
unsigned char far *zeile,far *pz;
long s,z,Zeil,Spal,lz,Rest;

printf("Topfbild benennen\n");
strcpy(bildname,Bpfad);
strcat(bildname,ffblk.ff_name);
laenge=strlen(bildname);
strcpy(&bildname[laenge-5],"1.MED");
printf("Topfbildname (mediangefiltert): %s\n",bildname);

printf("Topfbild einlesen\n");
Schirm_positionieren(sb,zb);

if ((Dateibin=open(bildname,O_RDONLY|O_BINARY))==-1)
        {
        printf("\a\aError: Kann das Topfbild '%s' nicht öffnen!\a\a\n",bildname);
        exit(2);
        }

Spal=farcoreleft();
if((zeile=farmalloc(sizeof(unsigned char)*Spal))==NULL)
        {
        printf("\a\aError: Nicht genug Speicher für Topfbild!\a\a\n");
        close(Dateibin);
        exit(3);
        }
if (Spal>0xFFFF-FP_OFF(zeile))
        {
        Spal=0xFFFF-FP_OFF(zeile);
```

```
        }
farfree(zeile);

Zeil=La/Spal;
Rest=La%Spal;
lz=0L;

if (Zeil)
        {
        if((zeile=farmalloc(sizeof(unsigned char)*Spal))==NULL)
                {
                printf("\a\aError: Nicht genug Speicher für Topfbild!\a\a\n");
                close(Dateibin);
                exit(3);
                }
        for (z=0;z<Zeil;z++)
                {
                pz=zeile;
                read(Dateibin,zeile,sizeof(unsigned char)*(unsigned int)Spal);
                for (s=0;s<Spal;s++)
                        {
                        zi=(int)(lz/Sp);
                        si=(int)(lz%Sp);
                        write_Frame(si+sb,zi+zb,(int)*pz);
                        lz++;
                        pz++;
                        }
                }
        farfree(zeile);
        }

if (Rest)
        {
        if((zeile=farmalloc(sizeof(unsigned char)*Rest))==NULL)
                {
                printf("\a\aError: Nicht genug Speicher für Topfbild!\a\a\n");
                close(Dateibin);
                exit(3);
                }
        pz=zeile;
        read(Dateibin,zeile,sizeof(unsigned char)*(unsigned int)Rest);
        for (s=0;s<Rest;s++)
                {
                zi=(int)(lz/Sp);
                si=(int)(lz%Sp);
                write_Frame(si+sb,zi+zb,(int)*pz);
                lz++;
                pz++;
                }
        farfree(zeile);
        }

close(Dateibin);

printf("Topfbild Maximum berechnen\n");
for(zi=((int)Ze/3+zb);zi<(2*(int)Ze/3+zb);zi++)
        for(si=((int)Sp/3+sb);si<(2*(int)Sp/3+sb);si++)
                if((unsigned char)read_Frame(si,zi)>max)
                        max=(unsigned char)read_Frame(si,zi);
```

```
printf("Shading korrigieren (Maximum = %d)\n",max);
for(zi=0;zi<(int)Ze;zi++)
        for(si=0;si<(int)Sp;si++)
                if(read_Frame(si+sb,zi+zb)!=0)
                        Bild[(long)zi*Sp+(long)si]=(unsigned
char)((double)Bild[(long)zi*Sp+(long)si]*((double)max/(double)read_Frame(si+sb,zi+zb)));
                else
                        Bild[(long)zi*Sp+(long)si]=0;
}
```

7.2.Lagekorrektur

Auf die nachstehend abgedruckten Programme möchte ich nicht näher eingehen, da sie lediglich eine Umarbeitung der von Herrn Lenz entwickelten Programme darstellen. 'estpar' ermittelt die Verschiebungsparameter zweier Bilder zueinander. Das Programm 'parameter_verknuepfen' berechnet die Gesamtverschiebung aus zwei Verschiebungsparametersätzen. 'imtran' transformiert ein Bild gemäß den übergebenen Parametern.

```
int estpar(unsigned char huge *b2,char order,int thresh,int offs,int offx,int offy,double *dstvec);
/* Bestimmung der Lageparameter zweier Bilder zueinander; falls das Programm        */
/* ordnungsgemäß gelaufen ist, gibt es 0 zurück                                     */
/* Das zweite Bild ist im Bildkartenspeicher abgelegt                               */
/* b2              :Bild 1                                                          */
/* order :Ordnung der Verschiebung (0...3); hier: 2                                 */
/* thresh          :Grenze für Berechnug  (je kleiner, desto mehr Punkte            */
/*                  werden behandelt); hier: 2                                      */
/* offs            :Ausschnittsgroessenparameter (offs Pixel vom Rand weg); hier: 30 */
/* offx            :a priori Verschiebung in x-Richtung; hier: 0                    */
/* offy            :a priori Verschiebung in y-Richtung; hier: 0                    */
/* dstvec          :Ergebniswert : Lageparameter                                    */
/*                  (pos. Wert von dstvec[0]: verschoben nach links)                 */
/*                  (pos. Wert von dstvec[1]: verschoben nach oben)                  */

int estparf(unsigned char huge *srcim2,double *dstvec,char order,int thresh,long *startxy,long
*stopxy);
/* Unterprogramm zu estpar                                                          */
int solve1(double *a,double *r,char np);
/* Unterprogramm zu estparf                                                         */

long imtran(unsigned char huge *b1,char order,double *dstvec,int sb,int zb);
/* Transformiert ein Bild gemäß den übergebenen Lageparametern; das                 */
/* Programm gibt als Funktion die Anzahl der gültigen Pixel zurück                  */
/* Das transformierte Bild steht im Bildkartenspeicher                              */
/* b1              :zu transformierendes Bild                                       */
/* order :Ordnung der Verschiebung (0...3); hier: 2                                 */
/* dstvec          :Lageparameter                                                   */
/*                  (pos. Wert von dstvec[0]: Verschiebung nach rechts)             */
/*                  (pos. Wert von dstvec[1]: Verschiebung nach unten)              */
/* sb              :Spaltenoffset des transformierten Bildes im Bildkartenspeicher  */
/* zb              :Zeilenoffset des transformierten Bildes im Bildkartenspeicher   */

void parameter_verknuepfen(double *b,double *a);
/* Verknüpfen der Transformationsparameter                                          */
/* b               :Neu berechneten Transformationsparameter                        */
/* a               :Alte Transformationsparameter und Zielparameter                 */

/*************************************************************************/

/*======R. Lenz, 27.03.91=======================

        estpar.c

        estimates image transformation parameters

==============================================*/

int estpar(unsigned char huge *b2,char order,int thresh,int offs,int offx,int offy,double *dstvec)
{
```

```
int x,y;
char i,np;
long startxy[2], stopxy[2];
double dx, dy;


if((order < 0) || (order > 3))
        {
        printf("order 0: shift, 1: affine, 2: perspect, 3: quadratic\n");
        exit(1);
        }

startxy[0] = startxy[1] = (long)offs;
stopxy[0] = Sp - (long)offs;
stopxy[1] = Ze - (long)offs;


estparf(&b2[Sp*offy+offx],dstvec,order,thresh,startxy,stopxy);


        /* number of parameters */
switch (order)
        {
        case 0: np = 2; break;
        case 1: np = 6; break;
        case 2: np = 8; break;
        case 3: np =12; break;
        }
for(i=0; i< np; i += 2)
        printf("x: %10.6lf, y: %10.6lf\n",dstvec[i],dstvec[i+1]);
if(order)
        {
        printf("displacements in corners:\n");
        for(y=0;y<=(int)Ze;y+=(int)Ze)
                {
                for(x=0;x<=(int)Sp;x+=(int)Sp)
                        {
dx=dstvec[0]+dstvec[2]*x+dstvec[4]*y+dstvec[6]*x*y+dstvec[8]*x*x+dstvec[10]*y*y;
dy=dstvec[1]+dstvec[3]*x+dstvec[5]*y+dstvec[7]*x*y+dstvec[9]*x*x+dstvec[11]*y*y;
                        printf("x: %10.6lf, y: %10.6lf      ",dx,dy);
                        }
                printf("\n");
                }
        } /* end if(order) */
return(0);
}

/*************************************************************************/

/*
/@ Short-title: estimates parameter of geometric image transformation
/@
/@ ============================================
/@
/@
/@      ALGORITHM
/@
/@      x = x' + dstvec[0]                      : order 0
/@            + dstvec[2]x'   + dstvec[4]y'         : order 1
```

```
/@            + dstvec[6]x'y'                    : order 2
/@            + dstvec[8]x'x' + dstvec[10]y'y'       : order 3
/@
/@      y = y' + dstvec[1]
/@            + dstvec[3]x'   + dstvec[5]y'
/@            + dstvec[7]x'y'
/@            + dstvec[9]x'x' + dstvec[11]y'y'
/@
/@
/@      x ,y  coordinates sourceimage1
/@      x',y' coordinates sourceimage2
/@
/@
/@
*************************************************************************/

int estparf(unsigned char huge *srcim2,double *dstvec,char order,int thresh,long *startxy,long
*stopxy)
/*
                                    first image ist im Bildspeicher abgelegt
unsigned char huge *srcim2;     second image
double *dstvec;             output parameter vector
int thresh;                         threshhold
char order;                         transformation type
long *startxy, *stopxy;     window origins, ends(not within window)
*/
{
long sizex;                         /* size in x-dimension, number of windows */
register long ix;           /* image index */
register long x,y;                  /* image coordinates  */
register long sx,sy;                /* average gradient  */
register double den;                /* lenght of average gradient vector */
register long st;           /* temporal image gradient */
register double stden;              /* product of st and den   */
long sxi,syi,sxo,syo;               /* spatial image gradients */
char np;                            /* number of parameters,depends on order */
double p[MAXPAR];                   /* parameter vector */
double a[78];                       /* symmetric positiv definite matrix A */
                                    /* holding the sum over gradient products */
                                    /* is inverted to give resulting vector p */
                                    /* a[0],a[1],...a[77] =
                                       A(1,1)
                                       A(2,1) A(2,2)
                                       A(3,1) A(3,2) A(3,3)
                                        ....
                                                        ... A(12,12)    */
int  i,k,in;                        /* array indices */
double q[MAXPAR];                   /* holds gradient-coordinate products */
long count = 0;             /* anzahl der berechneten punkte */
int s1,s2,z1,z2;
  /* Initialisierungen */

thresh = 2 * thresh * thresh;
sizex=Sp;
switch( order )
          {
          case 0 : np = 2; break;
          case 1 : np = 6; break;
          case 2 : np = 8; break;
```

```
        case 3 : np = 12;break;
        }

for( i = in = 0; i < np; i++)
        {
        p[i] = 0.0;
        for( k = 0; k <= i; k++)
                a[in++] = 0.0;
        }

  /* Ende Initialisierungen */

for( y = startxy[1]; y < stopxy[1]; y++ )
        {
        ix = sizex*y + startxy[0]; /* index of center pixel */
        for( x = startxy[0]; x < stopxy[0]; x++ )
                {
                z1=(int)((ix+1)/Sp);
                s1=(int)((ix+1)-(long)z1*Sp);
                z2=(int)((ix-1)/Sp);
                s2=(int)((ix-1)-(long)z2*Sp);
                sxi = (0xff&(unsigned char)read_Frame(s1,z1)) - (0xff&(unsigned
char)read_Frame(s2,z2));

                z1=(int)((ix+sizex)/Sp);
                s1=(int)((ix+sizex)-(long)z1*Sp);
                z2=(int)((ix-sizex)/Sp);
                s2=(int)((ix-sizex)-(long)z2*Sp);
                syi = (0xff&(unsigned char)read_Frame(s1,z1)) - (0xff&(unsigned
char)read_Frame(s2,z2));

                sxo = (0xff&srcim2[ix+1]) - (0xff&srcim2[ix-1]);
                syo = (0xff&srcim2[ix+sizex]) - (0xff&srcim2[ix-sizex]);

                if(sxi*sxo + syi*syo > thresh )
                        {
                        count++;                                /* increment count */
                        sx = sxi + sxo;             /* average gradient */
                        sy = syi + syo;
                                /* denominator */
                        den = 1.0/(sx*sx + sy*sy);
                                /* den = 1.0; */

                        z1=(int)((ix)/Sp);
                        s1=(int)((ix)-(long)z1*Sp);

                        st = 4*((0xff&srcim2[ix]) - (0xff&(unsigned char)read_Frame(s1,z1)));
                        stden = st * den;

                        q[0] = sx;
                        q[1] = sy;
                        if( order > 0 )             /* calculate grad.-coord.products */
                                {
                                q[2] = sx*x;
                                q[3] = sy*x;
                                q[4] = sx*y;
                                q[5] = sy*y;
                                if( order > 1 )
                                        {
```

```
                                        q[6] = q[2] * y;
                                        q[7] = q[3] * y;
                                        if( order > 2 )
                                                {
                                                q[8]  = q[2] * x;
                                                q[9]  = q[3] * x;
                                                q[10] = q[4] * y;
                                                q[11] = q[5] * y;
                                                }
                                        }
                                }
                        for(i=in=0;i<np;i++)
                                {
                                p[i] += q[i] * stden;
                                for(k=0;k<=i;k++)
                                a[in++] += q[i] * q[k] * den;
                                }
                        }
                ix++;   /* increment pointer */
                }       /* end x-loop */
        }       /* end y-loop */

if(solve1(a,p,np) != 0)             /* solve linear equation system A*x = p */
        {                           /* x is returned in p */
        printf("Matrixinversion irregulaer\n");
        return(-6);
        }

for(i=0;i<np;i++)
        dstvec[i] = p[i];

for(i=np;i<MAXPAR;i++)
        dstvec[i] = 0.0;

printf("used dots: %ld\n",count);

return(0);
}

/*************************************************************************/

/* bvm/reimar/solve1.c */
/* solves a system of linear equations Ax = r with symmetric positive
   definite matrix A */

int solve1(double *a,double *r,char np)
/*
double *a;          lower half of matrix A, stored in a one-dimensional array
                                        storage mode:
                                        a[0]
                                        a[1]  a[2]
                                        a[3]  a[4]  a[5]
                                        a[6]  a[7]  a[8]  etc.
double *r;          right hand side of linear equation system,
                                        on return solution vector
char np;            dimension of r and matrix A
*/
{
int j,k,l;  /* running variables */
```

```
int ib,in,kl;       /* array indices */
double sum;     /* intermediate results */


for(k=in=0;k<np;k++)            /* bring A into cholesky factorized form */
        {
        sum = a[ib=in];
        for(j=kl=0;j<k;j++)       /* falls through for k==0 */
                {
                a[in++] = sum/a[kl++];
                sum = a[in];
                for(l=ib;l<in;l++) sum -= a[kl++]*a[l];
                }
        if(sum<0)
                {
                printf("solve: subdeterminant too small\n");
                a[in] = sum = 0;
                }
        else a[in] = sqrt(sum);
        in++;
        }

for(k=kl=0;k<np;k++)                /* solve for x, part 1 */
        {
        sum = r[k];
        for(l=0;l<k;l++) sum -= a[kl++]*r[l];
        r[k] = sum/a[kl++];
        }

        /* k = np */
while(k--)          /* part 2 */
        {
        sum = r[k];
        in = kl + k;
        for(l=k+1;l<np;l++)
                {
                sum -= a[in++]*r[l];
                in += l;
                }
        r[k] = sum/a[--kl];
        kl -= k;
        }

return(0);
}

/*****************************************************************************/

/*======R. Lenz, 2.04.91==============================

        imtran.c

        geometric transformation of an image
*/
/* ***************************************************************************
/@ imtran.c
/@
/@      ALGORITHM
```

```
/@
/@      x',y' = coordinates of srcim
/@      x,y   = coordinates of dstim
/@
/@
/@      x = x' + srcvec[0]                      : order 0     image shift only
/@            + srcvec[2]x'   + srcvec[4]y'       : order 1     + affine transf.
/@            + srcvec[6]x'y'                   : order 2     + bilinear transf.
/@            + srcvec[8]x'x' + srcvec[10]y'y'    : order 3     + quadratic transf.
/@
/@      y = y' + srcvec[1]
/@            + srcvec[3]x'   + srcvec[5]y'
/@            + srcvec[7]x'y'
/@            + srcvec[9]x'x' + srcvec[11]y'y'
/@
*************************************************************************/

long imtran(unsigned char huge *b1,char order,double *dstvec,int sb,int zb)

{
register long xi,yi;                    /* input coordinates, srcimage  */
register long xi1,yi1;                  /* 1 + input coordinates */
register long xo,yo;                    /* output coordinates, dstimage */
double fxi,fyi,dx,dy,ddx,ddy;           /* input coordinates */
double frx,fry;             /* fractinal part of input coord. */
double frx1,fry1;           /* 1.0 - fract. part of input coord. */
double value;               /* ergebnis Interpolation  */
char i;
long anz=0;

for(i=0;i<12;i++) dstvec[i]=(-1.0)*dstvec[i];

if((order < 0) || (order > 3))
        {
        printf("order 0: shift, 1: affine, 2: perspect, 3: quadratic\n");
        exit(1);
        }

/*************************************************************************/

ddx = dstvec[8] * 2.0;
ddy = dstvec[9] * 2.0;

for( yo=0; yo<Ze; yo++ )
        {
        fxi = dstvec[0];            /* order 0 */
        fyi = yo + dstvec[1];
        dx=1.0;
        dy=0.0;
        switch(order)
                {
                case 3: fxi += dstvec[10] * yo * yo ;
                        fyi += dstvec[11] * yo * yo ;
                        dx += dstvec[8];
                        dy += dstvec[9];

                case 2: dx += dstvec[6] * yo ;
                        dy += dstvec[7] * yo ;
```

```
                case 1: fxi += dstvec[4] * yo ;
                        fyi += dstvec[5] * yo ;
                        dx += dstvec[2];
                        dy += dstvec[3];
                }

        for(xo=0;xo<Sp;xo++)
                {


                                        /*  7 | 8 | 1 */
                                        /*  --+---+-- */
                                        /*  6 | 0 | 2 */
                                        /*  --+---+-- */
                                        /*  5 | 4 | 3 */
                                        /* 0 Orginalbild */

                if(fxi < 0.0)
                        write_Frame((int)xo+sb,(int)yo+zb,0);
                else if(fxi > (double)(Sp-1))
                        write_Frame((int)xo+sb,(int)yo+zb,0);
                else
                        {
                        if(fyi < 0.0)                           /* 8 */
                                write_Frame((int)xo+sb,(int)yo+zb,0);
                        else if(fyi > (double)(Ze-1))
                                write_Frame((int)xo+sb,(int)yo+zb,0);
                        else                                    /* 0 */
                                {
                                anz++;
                                xi   = (long)fxi;
                                frx  = fxi - (double)xi;
                                frx1 = 1.0 - frx;
                                yi   = (long)fyi;
                                fry  = fyi - (double)yi;
                                fry1 = 1.0 - fry;
                                xi1  = xi+1;
                                yi1  = yi+1;
                                value = frx1 * fry1 * (double)b1[yi*Sp + xi]
                                     + frx  * fry1 * (double)b1[yi*Sp + xi1]
                                     + frx1 * fry  * (double)b1[yi1*Sp + xi]
                                     + frx  * fry  * (double)b1[yi1*Sp + xi1];
                                write_Frame((int)xo+sb,(int)yo+zb,(int)(value + 0.5));
                                }
                        }
                fxi += dx;
                fyi += dy;
                if (order > 2)
                        {
                        dx += ddx;
                        dy += ddy;
                        }
                }       /* end x loop  */
        }       /* end y loop  */

for(i=0;i<12;i++) dstvec[i]=(-1.0)*dstvec[i];
return(anz);
}
```

```
/*************************************************************************/

void parameter_verknuepfen(double *b,double *a)
{
int i;
double e[12];

e[0]=b[0]*(1.0+a[2]+a[6]*b[1])+a[0]+a[4]*b[1];
e[2]=a[2]+b[2]*(1.0+a[2]+a[6]*b[1])+a[4]*b[3]+a[6]*(b[1]+b[0]*b[3]);
e[4]=a[4]*(1.0+b[5])+b[4]*(1.0+a[2]+a[6]*b[1])+a[6]*b[0]*(1.0+b[5]);
e[6]=b[6]*(1.0+a[2])+a[4]*b[7]+a[6]*(b[5]+b[0]*b[7]+b[6]*b[1]+b[2]*(1.0+b[5]));
e[8]=0.0;
e[10]=0.0;

e[1]=b[1]*(1.0+a[5]+a[7]*b[0])+a[1]+a[3]*b[0];
e[3]=a[3]*(1.0+b[2])+b[3]*(1.0+a[5]+a[7]*b[0])+a[7]*b[1]*(1.0+b[2]);
e[5]=a[5]*(1.0+b[5])+b[4]*(a[3]+a[7]*b[1])+a[7]*b[0]*(1.0+b[5])+b[5];
e[7]=b[7]*(1.0+a[5])+a[3]*b[6]+a[7]*(b[0]*b[7]+b[6]*b[1]+b[4]*b[3]+(1.0*b[2])*(1.0+b[5]));
e[9]=0.0;
e[11]=0.0;

for(i=0;i<12;i++)
        a[i]=e[i];
}
```

7.3.Subtraktionsbild erstellen

Die Subtraktion zweier Bilder wird im Nachfolgenden Programm dargelegt. Subtraktionsergebnisse kleiner als Null werden zu Null gesetzt.

```
void bild_subtrahieren(unsigned char huge *Bild,int sb,int zb);
/* Subtraktion zweier Bilder voneinander; negative Werte werden zu Null          */
/* gesetzt; das Bild, das abgezogen wird, steht im Bildkartenspeicher            */
/* Bild          :Bild, von dem abgezogen wird, und Zielbild                     */
/* sb            :Spaltenoffset des zu subtrahierenden Bildes im Bildkartenspeicher  */
/* zb            :Zeilenoffset des zu subtrahierenden Bildes im Bildkartenspeicher   */

/*************************************************************************/

void bild_subtrahieren(unsigned char huge *Bild,int sb,int zb)
{
int z,s,graust;
for(z=0;z<(int)Ze;z++)
        {
        for(s=0;s<(int)Sp;s++)
                {
                graust=(int)Bild[(long)z*Sp+(long)s]-read_Frame(s+sb,z+zb);
                if (graust<0)
                        Bild[(long)z*Sp+(long)s]=0;
                else
                        Bild[(long)z*Sp+(long)s]=graust;
                }
        }
}
```

7.4.Medianfilterung

Dieses Programm kommt zweimal zur Anwendung. Zum einen werden damit die 'Topfbilder' gefiltert, zum anderen die Subtraktionsbilder. Beide Male wird ein 3*3 Kernel verwendet, d.h. 'kspalte' und 'kzeile' im Aufruf werden zu 3 gesetzt. Der ein Pixel breite Rand der Bilder wird dabei unbearbeitet übernommen. Zur Ermittelung des Medianwertes werden die acht direkten Nachbarn und der betroffene Punkt selbst in aufsteigender Reihenfolge sortiert in einen Vektor geschrieben. Der fünfte (mittlere) Wert im Vektor wird als neu berechneter Bildpunkt herangezgen. In Anbetracht der langwierigen Berechnung der Lagekorrektur wird hier auf einen schnelleren Algorithmus verzichtet.

```
void bild_medianfiltern(unsigned char huge *Bild,unsigned char kspalte,unsigned char kzeile,int
sb,int zb);
/* Medianfiltern eines Bildes; dabei wird der Rand nicht berücksichtigt     */
/* Das gefilterte Bild steht im Bildkartenspeicher                          */
/* Bild          :Zu filterndes Bild                                              */
/* kspalte       :Spaltenanzahl des filternden Kernels                            */
/* kzeile:Zeilenanzahl des filternden Kernels                             */
/* sb            :Spaltenoffset des gefilterten Bildes im Bildkartenspeicher      */
/* zb            :Zeilenoffset des gefilterten Bildes im Bildkartenspeicher*/

/*************************************************************************/

void bild_medianfiltern(unsigned char huge *Bild,unsigned char kspalte,unsigned char kzeile,int
sb,int zb)
{
int s,z,ks,kz,l;
unsigned char *Kernel;
unsigned char w,zw;

Kernel=calloc(kspalte*kzeile,sizeof(char));

for(z=0;z<(int)Ze;z++)
        {
        for(s=0;s<(int)Sp;s++)
                {
                if((s<(((int)kspalte-1)/2))||(z<(((int)kzeile-1)/2))||(s>(Sp-1-((int)kspalte-
1)/2))||(z>(Ze-1-((int)kzeile-1)/2)))
                        {
                        write_Frame(s+sb,z+zb,(int)Bild[(long)z*Sp+(long)s]);
                        }
                else
                        {
                        memset(Kernel,0,kzeile*kspalte);
                        for(kz=0;kz<(int)kzeile;kz++)
                                {
                                for(ks=0;ks<(int)kspalte;ks++)
                                        {
                                        w=Bild[(long)(z+kz)*Sp+(long)(s+ks)];
                                        for(l=0;l<((kz*kspalte+ks));l++)
                                                {
                                                if(w<Kernel[l])
                                                        {
                                                        zw=Kernel[l];
                                                        Kernel[l]=w;
                                                        w=zw;
                                                        }
                                                }
                                        Kernel[kz*(int)kspalte+ks]=w;
```

```
                                        }
                                    }
                                write_Frame(s+sb,z+zb,(int)Kernel[((int)kzeile*(int)kspalte-1)/2]);
                                }
                        }
                }
        free(Kernel);
        }
```

7.5.Unterdrückung des Grundrauschens

Hierbei werden die unteren fünf Grauwerte zu Null gesetzt, d.h. 'grenze' im Funktionsaufruf muß auf 6 gesetzt werden.

```
void bild_beschneiden(unsigned char huge *Bild,unsigned char grenze);
/* Grauwerte eines Bildes unterhalb 'grenze' zu Null setzen                    */
/* Bild             :zu beschneidendes Bild                                    */
/* grenze           :Grenze (wert<grenze --> 0)                                */

/*****************************************************************************/

void bild_beschneiden(unsigned char huge *Bild,unsigned char grenze)
{
long l;
for(l=0;l<La;l++)
        {
        if(Bild[l]<grenze)
                Bild[l]=0;
        }
}
```

7.6.Regionenabgrenzung

Das hier abgedruckte Programm kann zusammenhängende Pixel ungleich Null von einstellbarer Höchststreckenlänge in vier Richtungen aus dem Bild löschen. Zur Regionenabgrenzung werden 'z_grenz', 's_grenz', 'd1_grenz' und 'd2_grenz' jeweils zu 1 gesetzt. Zur Eliminierung dieser Streckenlängen wird das Bild horizontal, vertikal und in den zwei Diagonalrichtungen durchsucht. Sobald eine Strecke gefunden ist, die kleiner oder gleich der angegebenen Grenze ist, springt das Programm an die Anfangsstelle der Strecke zurück und löscht die Punkte.

```
void bild_strukturen_eliminieren(unsigned char huge *Bild,int z_grenz,int s_grenz,int
d1_grenz,int d2_grenz);
/* Eliminiert Strukturen != Null, die kleiner gleich ??grenz sind                  */
/* ( Ze > Sp !!!!!) Der 4. Quadrant des Bildkartenspeichers wird benutzt !         */
/* Bild          :zu behandelndes Bild und Zielbild                                */
/* z_grenz       :Zeilen-Grenzwert (Strukturen <= z_grenz => 0)                    */
/* s_grenz       :Spalten-Grenzwert (Strukturen <= s_grenz => 0)                   */
/* d1_grenz      :Diagonale 1-Grenzwert (Strukturen <= d1_grenz => 0)              */
/* d2_grenz      :Diagonale 2-Grenzwert (Strukturen <= d2_grenz => 0)              */

/*************************************************************************/

void bild_strukturen_eliminieren(unsigned char huge *bild,int z_grenz,int s_grenz,int
d1_grenz,int d2_grenz)
{
int s,z,l,struk[512][2];
char flag=0;
int laenge=0;

/*
z_grenz--;
s_grenz--;
d1_grenz--;
d2_grenz--;
*/

Schirm_positionieren(512,512); /* Die linke obere Ecke des  Monitors wird dabei auf de
angegebenen Punkt gesetzt; dient der Anzeige des Bildkartenspeicherinhalts. */

printf("Bild in den 4. Quadranten des Bildspeichers kopieren\n");
bildspeicher_schreiben(bild,512,512);

                /* Strukturen < Mediangrenze => eliminieren */
                /* Zeilen */

printf("Zeilen\n");

for(z=0;z<(int)Ze;z++)
        {
        if(flag==1)
                {
                flag=0;
                if(laenge<z_grenz)
                        {
                        for(l=0;l<=laenge;l++)
                                {
                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                }
                        }
                }
```

```
	for(s=0;s<(int)Sp;s++)
		{
		if(read_Frame(512+s,512+z)>0)
			{
			if(flag==1)
				{
				laenge++;
				struk[laenge][0]=s;
				struk[laenge][1]=z;
				}
			else
				{
				flag=1;
				laenge=0;
				struk[0][0]=s;
				struk[0][1]=z;
				}
			}
		else
			{
			if(flag==1)
				{
				flag=0;
				if(laenge<z_grenz)
					{
					for(l=0;l<=laenge;l++)
						{
						bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
						}
					}
				}
			}
		}
if(flag==1)
	{
	flag=0;
	if(laenge<z_grenz)
		{
		for(l=0;l<=laenge;l++)
			{
			bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
			}
		}
	}

		/* Spalten */

printf("Spalten\n");

for(s=0;s<(int)Sp;s++)
	{
	if(flag==1)
		{
		flag=0;
		if(laenge<s_grenz)
			{
			for(l=0;l<=laenge;l++)
				{
```

```
                                        bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                        }
                                }
                        }
                for(z=0;z<(int)Ze;z++)
                        {
                        if(read_Frame(512+s,512+z)>0)
                                {
                                if(flag==1)
                                        {
                                        laenge++;
                                        struk[laenge][0]=s;
                                        struk[laenge][1]=z;
                                        }
                                else
                                        {
                                        flag=1;
                                        laenge=0;
                                        struk[0][0]=s;
                                        struk[0][1]=z;
                                        }
                                }
                        else
                                {
                                if(flag==1)
                                        {
                                        flag=0;
                                        if(laenge<s_grenz)
                                                {
                                                for(l=0;l<=laenge;l++)
                                                        {
                                                        bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                                        }
                                                }
                                        }
                                }
                        }
        }
if(flag==1)
        {
        flag=0;
        if(laenge<s_grenz)
                {
                for(l=0;l<=laenge;l++)
                        {
                        bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                        }
                }
        }

                /* Diagonalen von links oben nach rechts unten */

printf("Diagonale von links oben nach rechts unten\n");

                /* obere Hälfte */
for(s=0;s<(int)Sp;s++)
        {
        if(flag==1)
                {
```

```
                flag=0;
                if(laenge<d1_grenz)
                        {
                        for(l=0;l<=laenge;l++)
                                {
                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                }
                        }
                }
        for(z=0;z<((int)Sp-s);z++)
                {
                if(read_Frame(512+s+z,512+z)>0)
                        {
                        if(flag==1)
                                {
                                laenge++;
                                struk[laenge][0]=s+z;
                                struk[laenge][1]=z;
                                }
                        else
                                {
                                flag=1;
                                laenge=0;
                                struk[0][0]=s+z;
                                struk[0][1]=z;
                                }
                        }
                else
                        {
                        if(flag==1)
                                {
                                flag=0;
                                if(laenge<d1_grenz)
                                        {
                                        for(l=0;l<=laenge;l++)
                                                {
                                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                                }
                                        }
                                }
                        }
                }
if(flag==1)
        {
        flag=0;
        if(laenge<d1_grenz)
                {
                for(l=0;l<=laenge;l++)
                        {
                        bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                        }
                }
        }

                /* untere Hälfte */

for(s=0;s<(int)Sp;s++)
        {
```

```
	if(flag==1)
		{
		flag=0;
		if(laenge<d1_grenz)
			{
			for(l=0;l<=laenge;l++)
				{
				bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
				}
			}
		}
	for(z=(s+(int)Ze-(int)Sp);z<(int)Ze;z++)
		{
		if(read_Frame(512+z-s-(int)Ze+(int)Sp,512+z)>0)
			{
			if(flag==1)
				{
				laenge++;
				struk[laenge][0]=z-s-(int)Ze+(int)Sp;
				struk[laenge][1]=z;
				}
			else
				{
				flag=1;
				laenge=0;
				struk[0][0]=z-s-(int)Ze+(int)Sp;
				struk[0][1]=z;
				}
			}
		else
			{
			if(flag==1)
				{
				flag=0;
				if(laenge<d1_grenz)
					{
					for(l=0;l<=laenge;l++)
						{
						bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
						}
					}
				}
			}
		}
if(flag==1)
	{
	flag=0;
	if(laenge<d1_grenz)
		{
		for(l=0;l<=laenge;l++)
			{
			bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
			}
		}
	}

		/* zwischen den Hälften */
```

```
for(z=1;z<((int)Ze-(int)Sp);z++)
        {
        if(flag==1)
                {
                flag=0;
                if(laenge<d1_grenz)
                        {
                        for(l=0;l<=laenge;l++)
                                {
                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                }
                        }
                }
        for(s=0;s<(int)Sp;s++)
                {
                if(read_Frame(512+s,512+s+z)>0)
                        {
                        if(flag==1)
                                {
                                laenge++;
                                struk[laenge][0]=s;
                                struk[laenge][1]=s+z;
                                }
                        else
                                {
                                flag=1;
                                laenge=0;
                                struk[0][0]=s;
                                struk[0][1]=s+z;
                                }
                        }
                else
                        {
                        if(flag==1)
                                {
                                flag=0;
                                if(laenge<d1_grenz)
                                        {
                                        for(l=0;l<=laenge;l++)
                                                {
                                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                                }
                                        }
                                }
                        }
                }
        }
if(flag==1)
        {
        flag=0;
        if(laenge<d1_grenz)
                {
                for(l=0;l<=laenge;l++)
                        {
                        bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                        }
                }
        }
```

```
                /* Diagonalen von rechts oben nach links unten */

printf("Diagonale von rechts oben nach links unten\n");

                /* obere Hälfte */
for(s=((int)Sp-1);s>=0;s--)
        {
        if(flag==1)
                {
                flag=0;
                if(laenge<d2_grenz)
                        {
                        for(l=0;l<=laenge;l++)
                                {
                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                }
                        }
                }
        for(z=0;z<=s;z++)
                {
                if(read_Frame(512+s-z,512+z)>0)
                        {
                        if(flag==1)
                                {
                                laenge++;
                                struk[laenge][0]=s-z;
                                struk[laenge][1]=z;
                                }
                        else
                                {
                                flag=1;
                                laenge=0;
                                struk[0][0]=s-z;
                                struk[0][1]=z;
                                }
                        }
                else
                        {
                        if(flag==1)
                                {
                                flag=0;
                                if(laenge<d2_grenz)
                                        {
                                        for(l=0;l<=laenge;l++)
                                                {
                                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                                }
                                        }
                                }
                        }
                }
        }
if(flag==1)
        {
        flag=0;
        if(laenge<d2_grenz)
                {
                for(l=0;l<=laenge;l++)
                        {
```

```
                        bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                        }
                }
        }

                /* untere Hälfte */

for(s=((int)Sp-1);s>=0;s--)
        {
        if(flag==1)
                {
                flag=0;
                if(laenge<d2_grenz)
                        {
                        for(l=0;l<=laenge;l++)
                                {
                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                }
                        }
                }
        for(z=((int)Ze-s-1);z<(int)Ze;z++)
                {
                if(read_Frame(512+(int)Sp+(int)Ze-2-s-z,512+z)>0)
                        {
                        if(flag==1)
                                {
                                laenge++;
                                struk[laenge][0]=(int)Ze+(int)Sp-2-s-z;
                                struk[laenge][1]=z;
                                }
                        else
                                {
                                flag=1;
                                laenge=0;
                                struk[0][0]=(int)Ze+(int)Sp-2-s-z;
                                struk[0][1]=z;
                                }
                        }
                else
                        {
                        if(flag==1)
                                {
                                flag=0;
                                if(laenge<d2_grenz)
                                        {
                                        for(l=0;l<=laenge;l++)
                                                {
                                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                                }
                                        }
                                }
                        }
                }
        }
if(flag==1)
        {
        flag=0;
        if(laenge<d2_grenz)
                {
```

```
                for(l=0;l<=laenge;l++)
                        {
                        bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                        }
                }
        }

                /* zwischen den Hälften */

for(z=1;z<((int)Ze-(int)Sp);z++)
        {
        if(flag==1)
                {
                flag=0;
                if(laenge<d2_grenz)
                        {
                        for(l=0;l<=laenge;l++)
                                {
                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                }
                        }
                }
        for(s=((int)Sp-1);s>=0;s--)
                {
                if(read_Frame(512+s,512+z-s-1+(int)Sp)>0)
                        {
                        if(flag==1)
                                {
                                laenge++;
                                struk[laenge][0]=s;
                                struk[laenge][1]=z+(int)Sp-s-1;
                                }
                        else
                                {
                                flag=1;
                                laenge=0;
                                struk[0][0]=s;
                                struk[0][1]=z+(int)Sp-s-1;
                                }
                        }
                else
                        {
                        if(flag==1)
                                {
                                flag=0;
                                if(laenge<d2_grenz)
                                        {
                                        for(l=0;l<=laenge;l++)
                                                {
                                                bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                                                }
                                        }
                                }
                        }
                }
        }
if(flag==1)
        {
        flag=0;
```

```
        if(laenge<d2_grenz)
                {
                for(l=0;l<=laenge;l++)
                        {
                        bild[(long)struk[l][1]*Sp+(long)struk[l][0]]=0;
                        }
                }
        }
Schirm_positionieren(0,0);
}
```

7.7.Regioneneinteilung

Nachfolgend sind die Programme zur Regioneneinteilung und zur Randextraktion aufgeführt. Zu beachten ist dabei, daß nicht mehr als 255 Regionen pro Bild erzeugt werden sollten, da jede Region durch einen entsprechenden Grauwert charakterisiert ist. Schon bei der Einteilung in Regionen werden deren Fläche, deren mittlere und deren maximale Intensität gemessen. Die Einheit der Flächen ist 100 µm². Als minimale Regionengröße haben sich 10 Pixel bewährt. Das Prinzip der Regioneneinteilung ist folgendes: Das Bild wird zeilenweise von links oben nach rechts unten nach einem Punkt ungleich Null durchsucht. Wird solch ein Punkt gefunden, wird im Programm 'objekt_fuellen' dieser als Startpunkt einer Kette gesetzt. Diese Kette wird so erzeugt, daß alle Punkte die ungleich Null sind und in der Vierer-Nachbarschaft des betrachteten Punktes liegen, neu an die Kette angehängt werden, falls diese Punkte noch nicht Bestandteil der Kette sind. Die Kette wird so Punkt für Punkt durchlaufen bis das Ende erreicht ist. Dadurch, daß bei der Abarbeitung der Kette jeder betrachtete Punkt im neuen Bild gesetzt wird, ist leicht zu erkennen, welcher Punkt bereits in die Kette aufgenommen wurde. Somit ist jeder Punkt der Kette auch Punkt einer zusammenhängenden Region, und bei der punktweisen Abarbeitung kann gleich die mittlere und die maximale Intensität der Region festgestellt werden. Die Fläche erhält man durch Zählen der Punkte und durch Multiplizieren mit der realen Fläche eines Punktes. Ist die Fläche kleiner, als die erlaubte, so wird durch das Programm 'objekt_loeschen' die Region nach dem selben Prinzip wieder gelöscht. Die Ränder der Regionen werden folgendermaßen erzeugt: Ist ein Punkt ungleich Null und hat dieser Punkt in seiner Vierer-Nachbarschaft einen Punkt gleich Null, so ist es ein Randpunkt einer Region. Punkte, die ganz am Rand des Bildes liegen, sind, sofern sie ungleich Null sind, automatisch Randpunkte einer Region. Zu den Vierer-Nachbarschaftspunkten gehören der obere, untere, linke und der rechte angrenzende Punkt.

```
int objekt_flaechen(unsigned char huge *Bild,int grenz,double *erg,double *mitt,char *max,int
bs,int bz);
/* Funktion, die das Bild in Regionen einteilt; jeder Region wird ein              */
/* anderer Grauwert zugewiesen; die Anzahl der Regionen wird zurückge-             */
/* geben, dabei werden erst ab einer bestimmten Pixel-Flächen-Grenze  */
/* die Regionen gewertet; gleichzeitig wird die Fläche, der mittlere               */
/* Grauwert und der maximale Grauwert einer jeden Region ermittelt; das            */
/* ermittelte Bild steht im Bildkartenspeicher                                     */
/* Bild          : zu behandelndes Bild                                            */
/* grenz : Pixel-Flächen-Grenze                                    */
/* erg           : Ergebnisvektor (reale Flächen)                                  */
/* mitt          : Ergebnisvektor (mittlere Intensität)                            */
/* max           : Ergebnisvektor (maximale Intensität)                            */
/* bs            : Spaltenoffset des neuen Bildes im Bildkartenspeicher  */
/* bz            : Zeilenoffset des neuen Bildes im Bildkartenspeicher             */

long objekt_fuellen(unsigned char huge *Bild,int wert,double *mitt,char *maxi,int s,int z,int bs,int
bz);
/* Funktion, die die Regionen mit einem festen Wert im Bildkarten-                 */
/* speicher anlegt, und die Anzahl der Pixel pro Region zurückgibt;                */
/* außerdem wird die mittlere und die maximale Intensität berechnet                */
/* Bild          : zu behandelndes Bild                                            */
/* wert          : Nummer der Region (Grauwerte werden damit gesetzt)              */
/* mitt          : Ergebnis (mittlere Intensität)                                  */
/* maxi          : Ergebnis (maximale Intensität)                  */
/* s             : Anfangskoordinate (Spalte) der Region                           */
/* z             : Anfangskoordinate (Zeile) der Region            */
/* bs            : Spaltenoffset des neuen Bildes im Bildkartenspeicher  */
/* bz            : Zeilenoffset des neuen Bildes im Bildkartenspeicher             */

void objekt_loeschen(int wert,int s,int z,int bs,int bz);
/* Löscht die Regionen, die zu klein sind, im Bildkartenspeicher   */
/* wert          : Nummer der Region, die gelöscht werden soll     */
```

```
/* s            : Anfangskoordinate (Spalte) der Region                    */
/* z            : Anfangskoordinate (Zeile) der Region                     */
/* bs           : Spaltenoffset des Bildes im Bildkartenspeicher           */
/* bz           : Zeilenoffset des Bildes im Bildkartenspeicher            */

void objekt_raender_erzeugen(unsigned char huge *Bild,int bs,int bz);
/* Extrahiert die Ränder der Regionen; das Ergebnisbild liegt im          */
/* Bildkartenspeicher                                                     */
/* Bild         : zu behandelndes Bild                                    */
/* bs           : Spaltenoffset des Zielbildes im Bildkartenspeicher      */
/* bz           : Zeilenoffset des Zielbildes im Bildkartenspeicher       */

/************************************************************************/

int objekt_flaechen(unsigned char huge *Bild,int grenz,double *erg,double *mitt,char *max,int
bs,int bz)
{
int s,z,anz=0,sum_anz=0;
long flaeche;
char maxi;
double mittel;

printf("Bildbereich löschen\n");

for(z=0;z<(int)Ze;z++)
        for(s=0;s<(int)Sp;s++)
                write_Frame(s+bs,z+bz,0);

printf("Objekte berechnen\n");

for(z=0;z<(int)Ze;z++)
        for(s=0;s<(int)Sp;s++)
                if((Bild[(long)z*Sp+(long)s]!=0)&&(read_Frame(s+bs,z+bz)==0))
                        {
                        anz++;
                        if(anz==256)
                                {
                                printf("\a\aBei (s,z)=(%d,%d) tritt ein Überlauf auf!\a\a\n",s,z);
                                anz=1;
                                sum_anz+=255;
                                }
                        flaeche=objekt_fuellen(Bild,anz,&mittel,&maxi,s,z,bs,bz);
                        if(flaeche<=grenz)
                                {
                                objekt_loeschen(anz,s,z,bs,bz);
                                anz--;
                                }
                        else
                    {
                                mitt[sum_anz+anz]=mittel;
                                max[sum_anz+anz]=maxi;
                                erg[sum_anz+anz]=(double)flaeche*pixelbreite*pixelhoehe;
                                /*
                                printf("Objekt Nr. %d:\tPixel-Fläche = %ld\tFläche =
%lf\n",(sum_anz+anz),flaeche,erg[sum_anz+anz]);
                                */
                                }
                        }
return(sum_anz+anz);
```

```
}

/*************************************************************************/

long objekt_fuellen(unsigned char huge *Bild,int wert,double *mitt,char *maxi,int s,int z,int bs,int
bz)
{
struct Pixel
        {
        int sp;
        int ze;
        };
static struct Pixel p[10000];
const int MAX=10000;
long sum_flaeche=1L,sum_pix=0L;
unsigned int flaeche=0,anfang=0,i;
char pix1,pix2,pix3,pix4;

p[0].sp=s;
p[0].ze=z;
write_Frame(s+bs,z+bz,wert);
sum_pix=Bild[(long)z*Sp+(long)s];
*maxi=Bild[(long)z*Sp+(long)s];
do
        {
        s=p[anfang].sp;
        z=p[anfang].ze;
        pix1=Bild[(long)(z-1)*Sp+(long)s];
        pix2=Bild[(long)(z+1)*Sp+(long)s];
        pix3=Bild[(long)z*Sp+(long)(s-1)];
        pix4=Bild[(long)z*Sp+(long)(s+1)];

        if((pix1!=0)&&(read_Frame(s+bs,z-1+bz)==0)&&(z>0))
                {
                sum_pix+=pix1;
                if(pix1>*maxi)
                        *maxi=pix1;
                flaeche++;
                p[flaeche].sp=s;
                p[flaeche].ze=z-1;
                write_Frame(s+bs,z-1+bz,wert);
                }
        if((pix2!=0)&&(read_Frame(s+bs,z+1+bz)==0)&&(z<((int)Ze-1)))
                {
                sum_pix+=pix2;
                if(pix2>*maxi)
                        *maxi=pix2;
                flaeche++;
                p[flaeche].sp=s;
                p[flaeche].ze=z+1;
                write_Frame(s+bs,z+1+bz,wert);
                }
        if((pix3!=0)&&(read_Frame(s-1+bs,z+bz)==0)&&(s>0))
                {
                sum_pix+=pix3;
                if(pix3>*maxi)
                        *maxi=pix3;
                flaeche++;
                p[flaeche].sp=s-1;
```

```
                p[flaeche].ze=z;
                write_Frame(s-1+bs,z+bz,wert);
                }
        if((pix4!=0)&&(read_Frame(s+1+bs,z+bz)==0)&&(s<((int)Sp-1)))
                {
                sum_pix+=pix4;
                if(pix4>*maxi)
                        *maxi=pix4;
                flaeche++;
                p[flaeche].sp=s+1;
                p[flaeche].ze=z;
                write_Frame(s+1+bs,z+bz,wert);
                }
        anfang++;
        if(flaeche>MAX-5)
                {
                for(i=anfang;i<=flaeche;i++)
                        p[i-anfang]=p[i];
                sum_flaeche+=(long)anfang;
                flaeche-=anfang;
                anfang=0;
                if(flaeche>MAX-5)
                        {
                        printf("\aWarnung: Vektor zu klein beim Füllen von Objekt Nr. %d
!!\a\n",wert);
                        }
                }
        }
while((anfang-1)!=flaeche);
*mitt=(double)sum_pix/(double)((long)flaeche+sum_flaeche);
return((long)flaeche+sum_flaeche);
}

/*************************************************************************/

void objekt_loeschen(int wert,int s,int z,int bs,int bz)
{
struct Pixel
        {
        int sp;
        int ze;
        };
static struct Pixel p[100];
const int MAX=100;
unsigned int flaeche=0,anfang=0,i;

p[0].sp=s;
p[0].ze=z;
write_Frame(s+bs,z+bz,0);
do
        {
        s=p[anfang].sp;
        z=p[anfang].ze;
        if((read_Frame(s+bs,z-1+bz)==wert)&&(z>0))
                {
          flaeche++;
                p[flaeche].sp=s;
                p[flaeche].ze=z-1;
                write_Frame(s+bs,z-1+bz,0);
```

```
                }
        if((read_Frame(s+bs,z+1+bz)==wert)&&(z<((int)Ze-1)))
                {
          flaeche++;
                p[flaeche].sp=s;
                p[flaeche].ze=z+1;
                write_Frame(s+bs,z+1+bz,0);
                }
        if((read_Frame(s-1+bs,z+bz)==wert)&&(s>0))
                {
          flaeche++;
                p[flaeche].sp=s-1;
                p[flaeche].ze=z;
                write_Frame(s-1+bs,z+bz,0);
                }
        if((read_Frame(s+1+bs,z+bz)==wert)&&(s<((int)Sp-1)))
                {
          flaeche++;
                p[flaeche].sp=s+1;
                p[flaeche].ze=z;
                write_Frame(s+1+bs,z+bz,0);
                }
        anfang++;
        if(flaeche>MAX-5)
                {
                for(i=anfang;i<=flaeche;i++)
                        p[i-anfang]=p[i];
                flaeche-=anfang;
                anfang=0;
                if(flaeche>MAX-5)
                        {
                        printf("\aWarnung: Vektor zu klein beim Löschen von Objekt Nr. %d
!!\a\n",wert);
                        }
                }
        }
while((anfang-1)!=flaeche);
}

/*********************************************************************/

void objekt_raender_erzeugen(unsigned char huge *Bild,int bs,int bz)
{
int s,z;

for(z=0;z<(int)Ze;z++)
        for(s=0;s<(int)Sp;s++)
                if((s==0)||(s==(int)Sp-1)||(z==0)||(z==(int)Ze-1))
                        write_Frame(s+bs,z+bz,Bild[(long)z*Sp+(long)s]);
                else
                        if((Bild[(long)z*Sp+(long)s]!=0)&&
                          ((Bild[((long)z-1)*Sp+(long)s]==0)||
                           (Bild[(long)z*Sp+(long)s+1]==0)||
                           (Bild[((long)z+1)*Sp+(long)s]==0)||
                           (Bild[(long)z*Sp+(long)s-1]==0)))
                                write_Frame(s+bs,z+bz,Bild[(long)z*Sp+(long)s]);
                        else
                                write_Frame(s+bs,z+bz,0);
}
```

7.8.Umfang

Der Umfang wird mit folgendem Programm gemessen. Die Regionen müssen dabei noch als Flächen vorliegen. Pixelweise wird das Bild nach null-nichtnull-Übergängen durchsucht. Diese Übergänge werden entsprechend ihrer Berührstrecke in realen Längen aufaddiert. Die Einheit des Umfangs ist 10 µm.

```
int objekt_umfang_messen(unsigned char huge *Bild,double *umfang);
/* Funktion, die den Umfang der Regionen mißt; gibt die Anzahl der          */
/* Regionen zurück; das Bild muß in Flächenform vorliegen                   */
/* Bild         : zu behandelndes Bild                                      */
/* umfang       : Ergebnisvektor (realer Umfang)                            */

/*************************************************************************/

int objekt_umfang_messen(unsigned char huge *Bild,double *umfang)
{
int s,z,anz=0;

memset(umfang,0,sizeof(double)*256);


for(z=0;z<(int)Ze;z++)
        for(s=0;s<(int)Sp;s++)
                if(Bild[(long)z*Sp+(long)s]!=0)
                        {
                        if(Bild[(long)z*Sp+(long)s]>anz)
                anz=Bild[(long)z*Sp+(long)s];
                        if(((Bild[((long)z-1)*Sp+(long)s]==0)&&(z!=0))||(z==0))
                                umfang[Bild[(long)z*Sp+(long)s]]+=pixelbreite;
                        if(((Bild[(long)z*Sp+(long)s+1]==0)&&(s!=(int)Sp-1))||(s==(int)Sp-1))
                                umfang[Bild[(long)z*Sp+(long)s]]+=pixelhoehe;
                        if(((Bild[((long)z+1)*Sp+(long)s]==0)&&(z!=(int)Ze-1))||(z==(int)Ze-1))
                                umfang[Bild[(long)z*Sp+(long)s]]+=pixelbreite;
                        if(((Bild[(long)z*Sp+(long)s-1]==0)&&(s!=0))||(s==0))
                umfang[Bild[(long)z*Sp+(long)s]]+=pixelhoehe;
                        }
/*
for(s=1;s<=anz;s++)
        printf("Objekt Nr. %d:\tUmfang = %lf\n",s,umfang[s]);
*/
return(anz);
}
```

7.9.Rundheit

Um die Rundheit der Regionen mit nachfolgendem Programm messen zu können, muß das Bild in Konturenform vorliegen, d.h. von den Regionen dürfen nur noch die Konturen im Bild sichtbar sein. Das Unterprogramm 'objekt_achteck' berechnet die Seitenlängen des die Region umschließenden Achtecks. Dabei wird, ausgehend von einem vorher gefundenen Startpunkt, die Kontur der Region durch Suchen des im Uhrzeigersinn folgenden Nachbarschaftspunktes durchlaufen. Es werden dabei die acht Punkte gesucht, die soweit wie möglich oben, unten, links, rechts und auf den zu äußerst gelegenen 45-Grad-Geraden liegen. Daraus werden die Eckpunkte des Achtecks bestimmt, und hieraus wiederum, unter Berücksichtigung der realen Streckenlängen, die Längen der Achteckseiten. Als Nachbarschaftspunkte kommen die acht direkt anliegenden Punkte in Betracht.

```
int objekt_rundheit_messen(unsigned char huge *Bild,double *flaeche,double *rund);
/* Funktion, die die Rundheit der Regionen mißt; gibt die Anzahl          */
/* der Regionen zurück; das Bild muß in Ränderform vorliegen              */
/* Bild          : zu behandelndes Bild                                   */
/* flaeche       : Vektor, in dem die Flächengrößen der Regionen stehen   */
/* rund          : Ergebnisvektor                                         */

void objekt_achteck(unsigned char huge *Bild,long l,double *d);
/* Berechnet das zu einer Region gehörige umschließende Achteck; die      */
/* Region muß in Ränderform vorliegen; als Ergebnis werden die Längen     */
/* der Seitenkanten berechnet; d[0] ist obere waagrechte Strecke, dann    */
/* im Uhrzeigersinn                                                       */
/* Bild          : zu behandelndes Bild                                   */
/* l             : Anfangskoordinate der Region im 'Bild'-Vektor          */
/* d             : Ergebnisvektor (reale Längen)                          */

/************************************************************************/

int objekt_rundheit_messen(unsigned char huge *Bild,double *flaeche,double *rund)
{
int anz=1;
long l;
double d[8],d_sum,pi;

pi=acos(-1.0);

printf("Rundheit berechnen\n");

for(l=0;l<La;l++)
        if(Bild[l]==anz)
                {
                objekt_achteck(Bild,l,d);
                d_sum=d[0]+d[1]+d[2]+d[3]+d[4]+d[5]+d[6]+d[7];
                if(d_sum!=0.0)
                        rund[anz]=4.0*pi*flaeche[anz]/pow(d_sum,2.0);
                else
                        rund[anz]=0.0;
                anz++;
                }

return(anz-1);
}

/************************************************************************/

void objekt_achteck(unsigned char huge *Bild,long l,double *d)
```

```
{
int wert,e_p[2][8],r_p[8],s,z;
char pos_g=3,pos_a;
long l_lauf,l_vor;

r_p[0]=(int)Ze;
r_p[1]=(int)Ze+(int)Sp;
r_p[2]=0;
r_p[3]=0;
r_p[4]=0;
r_p[5]=0;
r_p[6]=(int)Sp;
r_p[7]=(int)Ze+(int)Sp;

wert=Bild[l];
l_vor=l;
do
        {
        pos_a=(pos_g+5)%8;
        switch (pos_a)
                {
                case 0:
                        l_lauf=l_vor+1L;
                        if(((l_lauf%Sp)!=0L)&&(Bild[l_lauf]==wert))
                                {
                                pos_g=0;
                                break;
                                }
                case 1:
                        l_lauf=l_vor+Sp+1L;
                        if(((l_lauf%Sp)!=0L)&&(l_lauf<La)&&(Bild[l_lauf]==wert))
                                {
                                pos_g=1;
                                break;
                                }
                case 2:
                        l_lauf=l_vor+Sp;
                        if((l_lauf<La)&&(Bild[l_lauf]==wert))
                                {
                                pos_g=2;
                                break;
                                }
                case 3:
                        l_lauf=l_vor+Sp-1L;
                        if(((l_lauf%Sp)!=(Sp-1L))&&(l_lauf<La)&&(Bild[l_lauf]==wert))
                                {
                                pos_g=3;
                                break;
                                }
                case 4:
                        l_lauf=l_vor-1L;
                        if(((l_lauf%Sp)!=(Sp-1L))&&(Bild[l_lauf]==wert))
                                {
                                pos_g=4;
                                break;
                                }
                case 5:
                        l_lauf=l_vor-Sp-1L;
                        if(((l_lauf%Sp)!=(Sp-1L))&&(l_lauf>=0L)&&(Bild[l_lauf]==wert))
```

```
                {
                pos_g=5;
    break;
                }
case 6:
        l_lauf=l_vor-Sp;
        if((l_lauf>=0L)&&(Bild[l_lauf]==wert))
                {
                pos_g=6;
                break;
                }
case 7:
        l_lauf=l_vor-Sp+1L;
        if(((l_lauf%Sp)!=0L)&&(l_lauf>=0L)&&(Bild[l_lauf]==wert))
                {
                pos_g=7;
                break;
                }
        l_lauf=l_vor+1L;
        if(((l_lauf%Sp)!=0L)&&(Bild[l_lauf]==wert))
                {
                pos_g=0;
                break;
                }
        l_lauf=l_vor+Sp+1L;
        if(((l_lauf%Sp)!=0L)&&(l_lauf<La)&&(Bild[l_lauf]==wert))
                {
                pos_g=1;
                break;
                }
        l_lauf=l_vor+Sp;
        if((l_lauf<La)&&(Bild[l_lauf]==wert))
                {
                pos_g=2;
                break;
                }
        l_lauf=l_vor+Sp-1L;
        if(((l_lauf%Sp)!=(Sp-1L))&&(l_lauf<La)&&(Bild[l_lauf]==wert))
                {
                pos_g=3;
                break;
                }
        l_lauf=l_vor-1L;
        if(((l_lauf%Sp)!=(Sp-1L))&&(Bild[l_lauf]==wert))
                {
                pos_g=4;
                break;
                }
        l_lauf=l_vor-Sp-1L;
        if(((l_lauf%Sp)!=(Sp-1L))&&(l_lauf>=0L)&&(Bild[l_lauf]==wert))
                {
                pos_g=5;
    break;
                }
        l_lauf=l_vor-Sp;
        if((l_lauf>=0L)&&(Bild[l_lauf]==wert))
                {
                pos_g=6;
                break;
```

```
                                        }
                                l_lauf=l_vor;
                                pos_g=8;
                                break;
                        }
                l_vor=l_lauf;
                s=(int)(l_lauf%Sp);
                z=(int)(l_lauf/Sp);
                if(z<r_p[0])
                        r_p[0]=z;
                if(s>r_p[2])
                        r_p[2]=s;
                if(z>r_p[4])
                        r_p[4]=z;
                if(s<r_p[6])
                        r_p[6]=s;
                if((s+z)<r_p[7])
                        r_p[7]=s+z;
                if((s+z)>r_p[3])
                        r_p[3]=s+z;
                if(((int)Sp-s+z)<r_p[1])
                        r_p[1]=(int)Sp-s+z;
                if(((int)Sp-s+z)>r_p[5])
                        r_p[5]=(int)Sp-s+z;
                }
        while(l!=l_lauf);

        e_p[0][0]=r_p[7]-r_p[0];
        e_p[1][0]=r_p[0];
        e_p[0][1]=(int)Sp-r_p[1]+r_p[0];
        e_p[1][1]=r_p[0];
        e_p[0][2]=r_p[2];
        e_p[1][2]=r_p[1]-(int)Sp+r_p[2];
        e_p[0][3]=r_p[2];
        e_p[1][3]=r_p[3]-r_p[2];
        e_p[0][4]=r_p[3]-r_p[4];
        e_p[1][4]=r_p[4];
        e_p[0][5]=(int)Sp-r_p[5]+r_p[4];
        e_p[1][5]=r_p[4];
        e_p[0][6]=r_p[6];
        e_p[1][6]=r_p[5]-(int)Sp+r_p[6];
        e_p[0][7]=r_p[6];
        e_p[1][7]=r_p[7]-r_p[6];

        d[0]=(double)(abs(e_p[0][1]-e_p[0][0]))*pixelbreite;
        d[2]=(double)(abs(e_p[1][3]-e_p[1][2]))*pixelhoehe;
        d[4]=(double)(abs(e_p[0][4]-e_p[0][5]))*pixelbreite;
        d[6]=(double)(abs(e_p[1][6]-e_p[1][7]))*pixelhoehe;
        d[1]=sqrt(pow((double)(e_p[0][2]-e_p[0][1])*pixelbreite,2.0)+
                pow((double)(e_p[1][2]-e_p[1][1])*pixelhoehe,2.0));
        d[3]=sqrt(pow((double)(e_p[0][3]-e_p[0][4])*pixelbreite,2.0)+
                pow((double)(e_p[1][4]-e_p[1][3])*pixelhoehe,2.0));
        d[5]=sqrt(pow((double)(e_p[0][5]-e_p[0][6])*pixelbreite,2.0)+
                pow((double)(e_p[1][5]-e_p[1][6])*pixelhoehe,2.0));
        d[7]=sqrt(pow((double)(e_p[0][0]-e_p[0][7])*pixelbreite,2.0)+
                pow((double)(e_p[1][7]-e_p[1][0])*pixelhoehe,2.0));
        }
```

7.10.Länglichkeit

Auch hier müssen die Regionen in Konturenform vorliegen. Das bei folgendem Programm benötigte Unterprogramm 'objekt_achteck' ist in Kapitel 7.9. beschrieben und abgedruckt.

```
int objekt_laenglichkeit_messen(unsigned char huge *Bild,double *laeng);
/* Funktion, die die Länglichkeit der Regionen mißt; gibt die Anzahl          */
/* der Regionen zurück; das Bild muß in Ränderform vorliegen                  */
/* Bild          : zu behandelndes Bild                                       */
/* laeng : Ergebnisvektor                                                     */

/****************************************************************************/

int objekt_laenglichkeit_messen(unsigned char huge *Bild,double *laeng)
{
int anz=1;
long l;
double d[8],e1,e2,e3,e4;

printf("Länglichkeit berechnen\n");

for(l=0;l<La;l++)
        if(Bild[l]==anz)
                {
                objekt_achteck(Bild,l,d);
                e1=d[0]+d[4];
                e2=d[1]+d[5];
                e3=d[2]+d[6];
                e4=d[3]+d[7];

                if(e1==0.0)
                        e1=pixelbreite;
                if(e2==0.0)
                        e2=sqrt(pow(pixelhoehe,2.0)+pow(pixelbreite,2.0));
                if(e3==0.0)
                        e3=pixelhoehe;
                if(e4==0.0)
                        e4=sqrt(pow(pixelhoehe,2.0)+pow(pixelbreite,2.0));

                laeng[anz]=e1/e3;
                if((e2/e4)>laeng[anz])
                        laeng[anz]=e2/e4;
                if((e3/e1)>laeng[anz])
                        laeng[anz]=e3/e1;
                if((e4/e2)>laeng[anz])
                        laeng[anz]=e4/e2;

                anz++;
                }

return(anz-1);
}
```

7.11.Verwinkeltheit

Zur Berechnung der Verwinkeltheit müssen die Regionen ebenfalls in Konturenform vorliegen. Das Programm 'objekt_winkel_messen' sucht das Bild zeilenweise von links oben nach links unten nach einem Punkt ungleich Null ab. Ist dieser Punkt in einem anfangs gelöschten Vergleichsbild ebenfalls ungleich Null, so wird das Unterprogramm 'objekt_winkel' gestartet. Der gefundene Punkt dient darin als Anfangspunkt zur Konturenverfolgung. Jeder gefundene Punkt wird im Vergleichsbild gesetzt, damit kein Punkt zweimal bearbeitet wird. Zur Konturenverfolgung sucht sich das Programm den - vom Vorgängerpunkt ausgehend - im Uhrzeigersinn folgenden Nachfolgepunkt, und bestimmt den so von dem Nachfolgepunkt und einem gedachten Nachfolgepunkt eingeschlossenen Winkel im Betrag. Der gedachte Nachfolgepunkt ist der, der in 180 Grad-Richtung folgen würde. Diese Winkel werden entlang der Kontur aufaddiert. Vorgänger- und Nachfolgepunkte sind aus der Menge der acht direkten Nachbarschaftspunkte des betrachteten Punktes.

```
int objekt_winkel_messen(unsigned char huge *Bild,double *winkel,int sb,int zb);
/* Funktion, die die Verwinkeltheit der Regionen mißt; gibt die Anzahl     */
/* der Regionen zurück; das Bild muß in Ränderform vorliegen; ein                */
/* Bereich des Bildkartenspeichers wird benötigt                           */
/* Bild          : zu behandelndes Bild                                    */
/* winkel        : Ergebnisvektor (realer Winkel)                          */
/* sb            : Spaltenoffset des Bildes im Bildkartenspeicher          */
/* zb            : Zeilenoffset des Bildes im Bildkartenspeicher                */

double objekt_winkel(unsigned char huge *Bild,long l,int sb,int zb);
/* Funktion, die die Verwinkeltheit einer Region berechnet; das Bild             */
/* muß in Ränderform vorliegen; ein Bereich des Bildkartenspeichers          */
/* wird benötigt                                                           */
/* Bild          : zu behandelndes Bild                                    */
/* l             : Anfangskoordinate der Region im 'Bild'-Vektor           */
/* sb            : Spaltenoffset des Bildes im Bildkartenspeicher          */
/* zb            : Zeilenoffset des Bildes im Bildkartenspeicher                */

/*************************************************************************/

int objekt_winkel_messen(unsigned char huge *Bild,double *winkel,int sb,int zb)
{
int anz=0;
long l;
int s,z;

memset(winkel,0,sizeof(double)*256);

printf("Bildbereich löschen\n");

for(z=0;z<(int)Ze;z++)
        for(s=0;s<(int)Sp;s++)
                write_Frame(s+sb,z+zb,0);

printf("Winkel berechnen\n");

for(l=0;l<La;l++)
        if((Bild[l]!=0)&&(read_Frame((int)(l%Sp)+sb,(int)(l/Sp)+zb)==0))
                {
                winkel[Bild[l]]+=objekt_winkel(Bild,l,sb,zb);
                if((int)Bild[l]>anz)
                        anz=(int)Bild[l];
                }
/*
```

```
for(s=1;s<=anz;s++)
        printf("Objekt Nr. %d:\tWinkel = %lf\n",s,winkel[s]);
*/
return(anz);
}

/*************************************************************************/

double objekt_winkel(unsigned char huge *Bild,long l,int sb,int zb)
{
int wert,anz=0;
char pos_g=3,pos_a,stop_flag=0;
long l_lauf=-1L,l_vor;
double winkel=0.0,w_sum,pi;
double winkel_spalte_diagonale;
double winkel_zeile_diagonale;

pi=acos(-1.0);
winkel_zeile_diagonale=atan(pixelhoehe/pixelbreite);
winkel_spalte_diagonale=pi/2.0-winkel_zeile_diagonale;
wert=Bild[l];
l_vor=l;
do
        {
        if(l==l_lauf)
                stop_flag=1;
        anz++;
        w_sum=0.0;
        pos_a=(pos_g+5)%8;
        switch (pos_a)
                {
                case 0:
                        l_lauf=l_vor+1L;
                        w_sum+=winkel_zeile_diagonale;
                        if(((l_lauf%Sp)!=0L)&&(Bild[l_lauf]==wert))
                                {
                                pos_g=0;
                                break;
                                }
                case 1:
                        l_lauf=l_vor+Sp+1L;
                        w_sum+=winkel_spalte_diagonale;
                        if(((l_lauf%Sp)!=0L)&&(l_lauf<La)&&(Bild[l_lauf]==wert))
                                {
                                pos_g=1;
                                break;
                                }
                case 2:
                        l_lauf=l_vor+Sp;
                        w_sum+=winkel_zeile_diagonale;
                        if((l_lauf<La)&&(Bild[l_lauf]==wert))
                                {
                                pos_g=2;
                                break;
                                }
                case 3:
                        l_lauf=l_vor+Sp-1L;
                        w_sum+=winkel_spalte_diagonale;
                        if(((l_lauf%Sp)!=(Sp-1L))&&(l_lauf<La)&&(Bild[l_lauf]==wert))
```

```
                    {
                    pos_g=3;
                    break;
                    }
case 4:
          l_lauf=l_vor-1L;
          w_sum+=winkel_zeile_diagonale;
          if(((l_lauf%Sp)!=(Sp-1L))&&(Bild[l_lauf]==wert))
                    {
                    pos_g=4;
                    break;
                    }
case 5:
          l_lauf=l_vor-Sp-1L;
          w_sum+=winkel_spalte_diagonale;
          if(((l_lauf%Sp)!=(Sp-1L))&&(l_lauf>=0L)&&(Bild[l_lauf]==wert))
                    {
                    pos_g=5;
     break;
                    }
case 6:
          l_lauf=l_vor-Sp;
          w_sum+=winkel_zeile_diagonale;
          if((l_lauf>=0L)&&(Bild[l_lauf]==wert))
                    {
                    pos_g=6;
                    break;
                    }
case 7:
          l_lauf=l_vor-Sp+1L;
          w_sum+=winkel_spalte_diagonale;
          if(((l_lauf%Sp)!=0L)&&(l_lauf>=0L)&&(Bild[l_lauf]==wert))
                    {
                    pos_g=7;
                    break;
                    }
          l_lauf=l_vor+1L;
          w_sum+=winkel_zeile_diagonale;
          if(((l_lauf%Sp)!=0L)&&(Bild[l_lauf]==wert))
                    {
                    pos_g=0;
                    break;
                    }
          l_lauf=l_vor+Sp+1L;
          w_sum+=winkel_spalte_diagonale;
          if(((l_lauf%Sp)!=0L)&&(l_lauf<La)&&(Bild[l_lauf]==wert))
                    {
                    pos_g=1;
                    break;
                    }
          l_lauf=l_vor+Sp;
          w_sum+=winkel_zeile_diagonale;
          if((l_lauf<La)&&(Bild[l_lauf]==wert))
                    {
                    pos_g=2;
                    break;
                    }
          l_lauf=l_vor+Sp-1L;
          w_sum+=winkel_spalte_diagonale;
```

```
                        if(((l_lauf%Sp)!=(Sp-1L))&&(l_lauf<La)&&(Bild[l_lauf]==wert))
                                {
                                pos_g=3;
                                break;
                                }
                        l_lauf=l_vor-1L;
                        w_sum+=winkel_zeile_diagonale;
                        if(((l_lauf%Sp)!=(Sp-1L))&&(Bild[l_lauf]==wert))
                                {
                                pos_g=4;
                                break;
                                }
                        l_lauf=l_vor-Sp-1L;
                        w_sum+=winkel_spalte_diagonale;
                        if(((l_lauf%Sp)!=(Sp-1L))&&(l_lauf>=0L)&&(Bild[l_lauf]==wert))
                                {
                                pos_g=5;
                                break;
                                }
                        l_lauf=l_vor-Sp;
                        w_sum+=winkel_zeile_diagonale;
                        if((l_lauf>=0L)&&(Bild[l_lauf]==wert))
                                {
                                pos_g=6;
                                break;
                                }
                        l_lauf=l_vor;
                        w_sum=0.0;
                        pos_g=8;
                        break;
                }
        l_vor=l_lauf;
        if(anz!=1)
                if(pi>w_sum)
                        winkel+=pi-w_sum;
                else
                        winkel+=w_sum-pi;
        write_Frame((int)(l_lauf%Sp)+sb,(int)(l_lauf/Sp)+zb,255);
        }
while(!stop_flag);

return(winkel);
}
```

7.12.Auswertung einzelner Kriterien

Die Beschreibungskriterien der Extravasationsregionen müssen spaltenweise in der Datei G:\DATEN\DATEN1.TXT abgespeichert sein. Über jeder Spalte sollte die Bezeichnung der Spalte stehen. Dasselbe gilt für die anderen Regionen. Hier heißt die Datei aber G:\DATEN\DATEN2.TXT.

Das folgende Programm trägt den Namen AUSWERT.C.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <alloc.h>
#include <conio.h>
#include <graphics.h>
#include <ctype.h>
#include <math.h>

void main();

void main()
{
FILE *Datei1;
FILE *Datei2;
char Dateiname1[25]={"G:\\DATEN\\DATEN1.TXT"};
char Dateiname2[25]={"G:\\DATEN\\DATEN2.TXT"};
char dummy[15],text1[15],text2[15],flag,key='A';
char flaglog=0,flagan,flagd,autom,flaglogv;
double ddummy,min,max,ber,help,dz1,dz2,minw,maxw;
double far *daten1,far *daten2;
double far *hist1,far *hist2;
double maxh1,maxh2,maxh3,maxh4,dp1,dp2,grenz,rpa,rpb;
int spal,i,j,anz1,z,p1,p2;
int anz2,is,iz1,iz2,izv1,izv2,iv;
int gdriver=DETECT,gmode;
int bz=319,bs=580;
long curpos,length;

/*************************************************************************/
printf("\n\nEingabe:\n");
/*************************************************************************/

printf("Die wievielte Spalte der Datendateien ? ");
scanf("%d",&spal);

/*************************************************************************/
printf("\nDaten 1 allociieren\n");
/*************************************************************************/

daten1=farcalloc(1000,sizeof(double));
if(daten1==NULL)
        {
        printf("\a\aError: Nicht genug Speicher um Daten 1 zu allociieren !\a\a\n");
        exit(1);
        }

/*************************************************************************/
printf("Daten 2 allociieren\n");
/*************************************************************************/
```

```
daten2=farcalloc(6000,sizeof(double));
if(daten2==NULL)
        {
        printf("\a\aError: Nicht genug Speicher um Daten 2 zu allociieren !\a\a\n");
        exit(1);
        }

/*************************************************************************/
printf("Histogramm 1 allociieren\n");
/*************************************************************************/

hist1=farcalloc(bs+1,sizeof(double));
if(hist1==NULL)
        {
        printf("\a\aError: Nicht genug Speicher um Histogramm 1  zu allociieren !\a\a\n");
        exit(1);
        }

/*************************************************************************/
printf("Histogramm 2 allociieren\n");
/*************************************************************************/

hist2=farcalloc(bs+1,sizeof(double));
if(hist2==NULL)
        {
        printf("\a\aError: Nicht genug Speicher um Histogramm 2 zu allociieren !\a\a\n");
        exit(1);
        }

/*************************************************************************/
printf("Daten 1 lesen\n");
/*************************************************************************/

if ((Datei1=fopen(Dateiname1,"rt"))==NULL)
        {
        printf("\a\aError: Kann die Datei '%s' nicht öffnen!\a\a\n",Dateiname1);
        exit(0);
        }

curpos=ftell(Datei1);
fseek(Datei1, 0L, SEEK_END);
length=ftell(Datei1);
fseek(Datei1, curpos, SEEK_SET);

for(i=1;i<=9;i++)
        {
        fscanf(Datei1,"%s",dummy);
        if(i==spal) strcpy(text1,dummy);
        }

curpos=ftell(Datei1);

j=0;
while(curpos<length)
        {
        for(i=1;i<=9;i++)
                {
                fscanf(Datei1,"%lf",&ddummy);
```

```
                    if(i==spal)
                              {
                              daten1[j]=ddummy;
                              j++;
                               }
                    }
          curpos=ftell(Datei1);
          }

anz1=j-1;

fclose(Datei1);


/*************************************************************************/
printf("Daten 2 lesen\n");
/*************************************************************************/

if ((Datei2=fopen(Dateiname2,"rt"))==NULL)
          {
          printf("\a\aError: Kann die Datei '%s' nicht öffnen!\a\a\n",Dateiname2);
          exit(0);
          }

curpos=ftell(Datei2);
fseek(Datei2, 0L, SEEK_END);
length=ftell(Datei2);
fseek(Datei2, curpos, SEEK_SET);

for(i=1;i<=9;i++)
          {
          fscanf(Datei2,"%s",dummy);
          if(i==spal) strcpy(text2,dummy);
          }

curpos=ftell(Datei2);

j=0;
while(curpos<length)
          {
          for(i=1;i<=9;i++)
                    {
                    fscanf(Datei2,"%lf",&ddummy);
                    if(i==spal)
                              {
                              daten2[j]=ddummy;
                              j++;
                              }
                    }
          curpos=ftell(Datei2);
          }

anz2=j-1;

fclose(Datei2);

/*************************************************************************/
printf("\nDatenausgabe:\n");
/*************************************************************************/
```

```
printf("Datenbezeichnung: %s;\tAnzahl der Daten 1: %i\n",text1,anz1);
printf("Datenbezeichnung: %s;\tAnzahl der Daten 2: %i\n",text2,anz2);


while(key!='E')
{
/*************************************************************************/
printf("\nEingaben:\n");
/*************************************************************************/

printf("\nAnzahl der Säulen (maximal: %d) ? ",bs);
scanf("%d",&z);
if (z<1) z=1;
if (z>bs) z=bs;

flaglogv=flaglog;
printf("Logarithmisch [J/(N)] ? ");
flaglog=toupper(getch());
printf("%c\n",flaglog);
if(flaglog!='J')
        flaglog=0;
else
        flaglog=1;

printf("Angleichen der Histogramme [(J)/N] ? ");
flagan=toupper(getch());
printf("%c\n",flagan);
if(flagan!='N')
        flagan=1;
else
        flagan=0;

printf("Drucken [J/(N)] ? ");
flagd=toupper(getch());
printf("%c\n",flagd);
if(flagd!='J')
        flagd=0;
else
        flagd=1;

/*************************************************************************/
/* Logarithmieren und Delogarithmieren der Daten */
/*************************************************************************/

if(!flaglogv && flaglog)
        {
        printf("\nLogarithmieren der Daten\n");
        for(i=0;i<anz1;i++)
                daten1[i]=log10(daten1[i]);
        for(i=0;i<anz2;i++)
                daten2[i]=log10(daten2[i]);
        }

if(flaglogv && !flaglog)
        {
        printf("\nDelogarithmieren der Daten\n");
        for(i=0;i<anz1;i++)
                daten1[i]=pow(10.0,daten1[i]);
```

```
        for(i=0;i<anz2;i++)
                daten2[i]=pow(10.0,daten2[i]);
        }

/*************************************************************************/
printf("\nBestimmung von Minimum und Maximum der Daten\n");
/*************************************************************************/

minw=300000.0;
maxw=-300000.0;
for(i=0;i<anz1;i++)
        {
        if(daten1[i]<minw) minw=daten1[i];
        if(daten1[i]>maxw) maxw=daten1[i];
        }

for(i=0;i<anz2;i++)
        {
        if(daten2[i]<minw) minw=daten2[i];
        if(daten2[i]>maxw) maxw=daten2[i];
        }

printf("Reales Minimum: %lf\tReales Maximum: %lf\n",minw,maxw);

printf("Neues Minimum (0 = Wert bestätigt) ? ");
scanf("%lf",&help);
if(help!=0.0)
        min=help;
else
        min=minw;

printf("Neues Maximum (0 = Wert bestätigt) ? ");
scanf("%lf",&help);
if(help!=0.0)
        max=help;
else
        max=maxw;

ber=(max-min)/(double)z;

printf("Minimum im Diagramm: %lf\tMaximum im Diagramm: %lf\nIntervallbreite:
%lf\n",min,max+ber,ber);

if(flagd)
        {
        fprintf(stdprn,"\n\nDatenbezeichnung: %s;\tAnzahl der Daten 1: %i\n",text1,anz1);
        fprintf(stdprn,"Datenbezeichnung: %s;\tAnzahl der Daten 2: %i\n",text2,anz2);
        fprintf(stdprn,"%d Säulen\t",z);
        if(!flaglog)
                fprintf(stdprn,"Lineare");
        else
                fprintf(stdprn,"Logarithmische");
        fprintf(stdprn," Darstellung\n");
        fprintf(stdprn,"Reales Minimum: %lf\tReales Maximum: %lf\n",minw,maxw);
        fprintf(stdprn,"Minimum im Diagramm: %lf\tMaximum im Diagramm: %lf\nIntervallbreite:
%lf\n",min,max+ber,ber);
        }

/*************************************************************************/
```

```
printf("\nBerechnung von Histogramm 1\n");
/*************************************************************************/

memset(hist1,0,sizeof(double)*(bs+1));
for(i=0;i<anz1;i++)
        {
        j=1;
        flag=0;
        while((!flag)&&(j<(z+2)))
                if(daten1[i]<=(min+(double)j*ber))
                        {
                        flag=1;
                        hist1[j-1]+=1.0;
                        }
                else
                        j++;
        }

/*************************************************************************/
printf("Berechnung von Histogramm 2\n");
/*************************************************************************/

memset(hist2,0,sizeof(double)*(bs+1));
for(i=0;i<anz2;i++)
        {
        j=1;
        flag=0;
        while((!flag)&&(j<(z+2)))
                if(daten2[i]<=(min+(double)j*ber))
                        {
                        flag=1;
                        hist2[j-1]+=1.0;
                        }
                else
                        j++;
        }

/*************************************************************************/
printf("Berechnung der Maxima der Histogramme\n");
/*************************************************************************/

maxh1=0.0;
maxh2=0.0;
for(i=0;i<=z;i++)
        {
        if(hist1[i]>maxh1) maxh1=hist1[i];
        if(hist2[i]>maxh2) maxh2=hist2[i];
        }
printf("Histogramm 1 (rot)   : Maximum = %d\n",(int)maxh1);
printf("Histogramm 2 (gruen) : Maximum = %d\n",(int)maxh2);
if(flagd)
        {
        fprintf(stdprn,"Histogramm 1 (rot)   : Maximum = %d\n",(int)maxh1);
        fprintf(stdprn,"Histogramm 2 (gruen) : Maximum = %d\n",(int)maxh2);
        if(!flagan)
                fprintf(stdprn,"Nicht a");
        else
                fprintf(stdprn,"A");
        fprintf(stdprn,"ngeglichene Histogramme\n");
```

```
        }

if(!flagan)
        {
        if(maxh1>maxh2)
                {
                maxh3=maxh1;
                maxh4=maxh1;
                }
        else
                {
                maxh3=maxh2;
                maxh4=maxh2;
                }
        }
else
        {
        printf("Angleichen der Histogramme\n");
        maxh3=maxh1;
        maxh4=maxh2;
        }

printf("\nZur Grafikausgabe beliebige Taste drücken!\n");
getch();

/*************************************************************************/
/* Grafikausgabe der Histogramme */
/*************************************************************************/

fflush(stdin);

initgraph(&gdriver,&gmode,"");

for (is=0;is<=bs;is++)
        putpixel(is,bz,15);

izv1=bz;
izv2=bz;
for (is=0;is<=bs;is++)
        {
        i=(int)(((double)is*(double)z/(double)bs)+0.5);

        iz1=bz-(int)((hist1[i]*(double)bz/maxh3)+0.5);
        iz2=bz-(int)((hist2[i]*(double)bz/maxh4)+0.5);

        if(iz1>izv1)
                for(j=izv1;j<=iz1;j++) putpixel(is,j,12);
        else
                for(j=izv1;j>=iz1;j--) putpixel(is,j,12);
        izv1=iz1;

        if(iz2>izv2)
                for(j=izv2;j<=iz2;j++)
                        if(getpixel(is,j)==12)
                                putpixel(is,j,11);
                        else
                                putpixel(is,j,10);
        else
                for(j=izv2;j>=iz2;j--)
```

```
                        if(getpixel(is,j)==12)
                                putpixel(is,j,11);
                        else
                                putpixel(is,j,10);
        izv2=iz2;
        }
getch();
closegraph();

/*************************************************************************/
/* Grafikausgabe der Häufigkeitsdichteverteilung */
/*************************************************************************/

fflush(stdin);

initgraph(&gdriver,&gmode,"");

for (is=0;is<=bs;is++)
        {
        putpixel(is,0,15);
        putpixel(is,(int)((double)bz/2.0+0.5),15);
        putpixel(is,bz,15);
        }

izv1=bz;
izv2=bz;
dz1=0.0;
dz2=0.0;
iv=-1;
for (is=0;is<=bs;is++)
        {
        i=(int)(((double)is*(double)z/(double)bs)+0.5);
        if(i!=iv)
                {
                dz1=dz1+hist1[i]*(double)bz/(double)anz1;
                dz2=dz2+hist2[i]*(double)bz/(double)anz2;
                iv=i;
                }

        iz1=bz-(int)(dz1+0.5);
        if(iz1>izv1)
                for(j=izv1;j<=iz1;j++) putpixel(is,j,12);
        else
                for(j=izv1;j>=iz1;j--) putpixel(is,j,12);
        izv1=iz1;

        iz2=bz-(int)(dz2+0.5);
        if(iz2>izv2)
                for(j=izv2;j<=iz2;j++)
                        if(getpixel(is,j)==12)
                                putpixel(is,j,11);
                        else
                                putpixel(is,j,10);
        else
                for(j=izv2;j>=iz2;j--)
                        if(getpixel(is,j)==12)
                                putpixel(is,j,11);
                        else
                                putpixel(is,j,10);
```

```
        izv2=iz2;
        }
getch();
closegraph();

/*************************************************************************/
printf("Grenzenauswertung:\n");
/*************************************************************************/

printf("\nZum Überspringen der Grenzenauswertung 'V' eingeben!\n");
key=toupper(getch());
while(key!='V')
        {
        printf("\nReales Minimum: %lf\tReales Maximum: %lf\n",minw,maxw);

        printf("\nAutomatische Auswertung [(J)/N] ? ");
        autom=toupper(getch());
        printf("%c\n",autom);
        if(autom!='N')
                {
                printf("Anzahl der Intervalle ? ");
                scanf("%d",&z);

                printf("Anfangswert (0 = Minimum bestätigt) ? ");
                scanf("%lf",&help);
                if(help!=0.0)
                        min=help;
                else
                        min=minw;

                printf("Endwert (0 = Maximum bestätigt) ? ");
                scanf("%lf",&help);
                if(help!=0.0)
                        max=help;
                else
                        max=maxw;

                ber=(max-min)/(double)z;

                printf("Anfangswert: %lf\tEndwert: %lf\tIntervallbreite: %lf\n",min,max,ber);
                printf("\n'1'<=\t'1'>\t'2'<=\t'2'>\t'p1'<=\t'p1'>\tals\n");
                if(flagd)
                        {
                        fprintf(stdprn,"\n\nAnfangswert: %lf\tEndwert: %lf\nIntervallbreite:
%lf\n",min,max,ber);
                        fprintf(stdprn,"\n'1'<=\t'1'>\t'2'<=\t'2'>\t'p1'<=\t'p1'>\tals\n");
                        }

                for(grenz=min;grenz<(max+ber/2.0);grenz+=ber)
                        {
                        p1=0;
                        for(i=0;i<anz1;i++)
                                if(daten1[i]<=grenz) p1++;

                        p2=0;
                        for(i=0;i<anz2;i++)
                                if(daten2[i]<=grenz) p2++;

                        dp1=(double)p1/(double)anz1*100.0;
```

```
                        dp2=(double)p2/(double)anz2*100.0;

                        if((p1+p2)!=0)
                                rpa=100.0*(double)p1/(double)(p1+p2);
                        else
                                rpa=999.0;

                        if((anz1-p1+anz2-p2)!=0)
                                rpb=100.0*(double)(anz1-p1)/(double)(anz1-p1+anz2-p2);
                        else
                                rpb=999.0;



        printf("%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%lf\n",dp1,1
00.0-dp1,dp2,100.0-dp2,rpa,rpb,grenz);
                        if(flagd)

        fprintf(stdprn,"%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%lf\
n",dp1,100.0-dp1,dp2,100.0-dp2,rpa,rpb,grenz);
                        }
                }
        else
                {
                printf("\nGrenze ? ");
                scanf("%lf",&grenz);

                p1=0;
                for(i=0;i<anz1;i++)
                        if(daten1[i]<=grenz) p1++;

                p2=0;
                for(i=0;i<anz2;i++)
                        if(daten2[i]<=grenz) p2++;

                dp1=(double)p1/(double)anz1*100.0;
                dp2=(double)p2/(double)anz2*100.0;

                if((p1+p2)!=0)
                        rpa=100.0*(double)p1/(double)(p1+p2);
                else
                        rpa=999.0;

                if((anz1-p1+anz2-p2)!=0)
                        rpb=100.0*(double)(anz1-p1)/(double)(anz1-p1+anz2-p2);
                else
                        rpb=999.0;


                printf("\n'1'<=\t'1'>\t'2'<=\t'2'>\t'p1'<=\t'p1'>\tals\n");

        printf("%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%lf\n",dp1,1
00.0-dp1,dp2,100.0-dp2,rpa,rpb,grenz);
                if(flagd)
                        {
                        fprintf(stdprn,"\n'1'<=\t'1'>\t'2'<=\t'2'>\t'p1'<=\t'p1'>\tals\n");

        fprintf(stdprn,"%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%3.2lf%%\t%lf\
n",dp1,100.0-dp1,dp2,100.0-dp2,rpa,rpb,grenz);
```

```
                        }
                }

        printf("\nZum Verlassen der Grenzenauswertung 'V' eingeben!\n");
        key=toupper(getch());
        }


printf("\nZum Beenden des Programms 'E' eingeben!\n");
key=toupper(getch());
}
farfree(hist2);
farfree(hist1);
farfree(daten2);
farfree(daten1);

}
```

7.13.ODER-Verknüpfung

Die Beschreibungskriterien der Extravasationsregionen müssen spaltenweise in der Datei G:\DATEN\DATEN1.TXT abgespeichert sein. Über jeder Spalte sollte die Bezeichnung der Spalte stehen. Dasselbe gilt für die anderen Regionen. Hier heißt die Datei aber G:\DATEN\DATEN2.TXT.

Das folgende Programm trägt den Namen ZUS_ODER.C.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

void main();

void main()
{
FILE *Datei;
char Dateiname[25]={"G:\\DATEN\\DATEN1.TXT"};
char Dateiname2[25]={"G:\\DATEN\\DATEN2.TXT"};
char dummy[15];
double daten[9],grenz50[9],grenz2[9],p50[2],p2[2],per2,per50;
int i,anz[2],a2[2],a50[2];
long curpos,length;
char k,key;


printf("\n\n\nODER-Verknüpfung\n================\n\n\n");

/*************************************************************************/
printf("\nDateneingabe:\n\n");
/*************************************************************************/

printf("\nExtravasationsregionenerkennung:\n\n");
printf("Untere Grenze für die Fläche (log) ? ");
scanf("%lf",&grenz50[0]);
if(grenz50[0]==0.0) grenz50[0]=3.15;
printf("Untere Grenze für das Maximum (log) ? ");
scanf("%lf",&grenz50[2]);
if(grenz50[2]==0.0) grenz50[2]=1.925;
printf("Untere Grenze für den Umfang (log) ? ");
scanf("%lf",&grenz50[4]);
if(grenz50[4]==0.0) grenz50[4]=2.8;
printf("Untere Grenze für die Verwinkeltheit (log) ? ");
scanf("%lf",&grenz50[8]);
if(grenz50[8]==0.0) grenz50[8]=2.44;
printf("Obere Grenze für Umfang/Fläche ? ");
scanf("%lf",&grenz50[5]);
if(grenz50[5]==0.0) grenz50[5]=0.2995;

printf("\n\nErkennung anderer Regionen:\n\n");
printf("Obere Grenze für die Fläche (log) ? ");
scanf("%lf",&grenz2[0]);
if(grenz2[0]==0.0) grenz2[0]=1.486895;
printf("Obere Grenze für den Mittelwert ? ");
scanf("%lf",&grenz2[1]);
```

```
if(grenz2[1]==0.0) grenz2[1]=12.613793;
printf("Obere Grenze für das Maximum (log) ? ");
scanf("%lf",&grenz2[2]);
if(grenz2[2]==0.0) grenz2[2]=1.220618;
printf("Obere Grenze für den Umfang (log) ? ");
scanf("%lf",&grenz2[4]);
if(grenz2[4]==0.0) grenz2[4]=1.430855;
printf("Untere Grenze für die Rundheit (log) ? ");
scanf("%lf",&grenz2[6]);
if(grenz2[6]==0.0) grenz2[6]=0.023674;
printf("Obere Grenze für die Verwinkeltheit (log) ? ");
scanf("%lf",&grenz2[8]);
if(grenz2[8]==0.0) grenz2[8]=0.911499;
printf("Untere Grenze für Umfang/Fläche ? ");
scanf("%lf",&grenz2[5]);
if(grenz2[5]==0.0) grenz2[5]=0.932414;
printf("\n\n");

/*************************************************************************/
/* Daten lesen */
/*************************************************************************/

for(k=0;k<2;k++)
        {
        if(k==1) strcpy(Dateiname,Dateiname2);
        printf("Daten %d lesen\n",k+1);
        if ((Datei=fopen(Dateiname,"rt"))==NULL)
                {
                printf("\a\aError: Kann die Datei '%s' nicht öffnen!\a\a\n",Dateiname);
                exit(0);
                }

        curpos=ftell(Datei);
        fseek(Datei, 0L, SEEK_END);
        length=ftell(Datei);
        fseek(Datei, curpos, SEEK_SET);

        for(i=1;i<=9;i++)
                fscanf(Datei,"%s",dummy);

        curpos=ftell(Datei);

        anz[k]=0;
        a50[k]=0;
        a2[k]=0;
        while(curpos<length)
                {
                for(i=0;i<9;i++)
                        fscanf(Datei,"%lf",&daten[i]);

                curpos=ftell(Datei);

                if(curpos<length)
                        {
                        daten[0]=log10(daten[0]);
                        daten[2]=log10(daten[2]);
                        daten[4]=log10(daten[4]);
                        daten[6]=log10(daten[6]);
                        daten[8]=log10(daten[8]);
```

```
                                anz[k]++;
                                if((daten[0]>grenz50[0])
                                 ||(daten[2]>grenz50[2])
                                 ||(daten[4]>grenz50[4])
                                 ||(daten[8]>grenz50[8])
                                 ||(daten[5]<=grenz50[5]))
                                        a50[k]++;

                                if((daten[0]<=grenz2[0])
                                 ||(daten[1]<=grenz2[1])
                                 ||(daten[2]<=grenz2[2])
                                 ||(daten[4]<=grenz2[4])
                                 ||(daten[6]>grenz2[6])
                                 ||(daten[8]<=grenz2[8])
                                 ||(daten[5]>grenz2[5]))
                                        a2[k]++;
                                }

                        }
                p50[k]=100.0*(double)a50[k]/(double)anz[k];
                p2[k]=100.0*(double)a2[k]/(double)anz[k];
                fclose(Datei);
                }
per2=100.0*(double)a2[1]/(double)(a2[0]+a2[1]);
per50=100.0*(double)a50[0]/(double)(a50[0]+a50[1]);

/*************************************************************************/
printf("\nErgebnisausgabe:\n");
/*************************************************************************/

printf("\nExtravasationsregionenerkennung:\n");
printf("Untere Grenze für die Fläche (log) : %lf\n",grenz50[0]);
printf("Untere Grenze für das Maximum (log) : %lf\n",grenz50[2]);
printf("Untere Grenze für den Umfang (log) : %lf\n",grenz50[4]);
printf("Untere Grenze für die Verwinkeltheit (log) : %lf\n",grenz50[8]);
printf("Obere Grenze für Umfang/Fläche : %lf\n",grenz50[5]);

printf("\nErkennung anderer Regionen:\n");
printf("Obere Grenze für die Fläche (log) : %lf\n",grenz2[0]);
printf("Obere Grenze für den Mittelwert : %lf\n",grenz2[1]);
printf("Obere Grenze für das Maximum (log) : %lf\n",grenz2[2]);
printf("Obere Grenze für den Umfang (log) : %lf\n",grenz2[4]);
printf("Untere Grenze für die Rundheit (log) : %lf\n",grenz2[6]);
printf("Obere Grenze für die Verwinkeltheit (log) : %lf\n",grenz2[8]);
printf("Untere Grenze für Umfang/Fläche : %lf\n",grenz2[5]);

printf("\n\nAnzahl der Extravasationsregionen: %d\n",anz[0]);
printf("Anzahl der anderen Regionen      : %d\n",anz[1]);
printf("\nErkannter Prozentsatz der Extravasationsregionen : %3.2lf%%\n",p50[0]);
printf("Dabei falsch erkannter Prozentsatz anderer Regionen : %3.2lf%%\n",p50[1]);
printf("Prozentsatz der Extravasationen von den Regionen,\ndie die Bedingung erfüllen :
%3.2lf%%\n",per50);
printf("\nErkannter Prozentsatz der anderen Regionen : %3.2lf%%\n",p2[1]);
printf("Dabei falsch erkannter Prozentsatz der Extravasationsregionen: %3.2lf%%\n",p2[0]);
printf("Prozentsatz der anderen Regionen von den Regionen,\ndie die Bedingung erfüllen :
%3.2lf%%\n",per2);

printf("\n\nDrucken [J/(N)] ? ");
```

```
key=toupper(getch());
printf("%c\n",key);
if(key=='J')
        {
        fprintf(stdprn,"\n\n\nODER-Verknüpfung\n================\n\n\n");
        fprintf(stdprn,"Extravasationsregionenerkennung:\n");
        fprintf(stdprn,"Untere Grenze für die Fläche (log) : %lf\n",grenz50[0]);
        fprintf(stdprn,"Untere Grenze für das Maximum (log) : %lf\n",grenz50[2]);
        fprintf(stdprn,"Untere Grenze für den Umfang (log) : %lf\n",grenz50[4]);
        fprintf(stdprn,"Untere Grenze für die Verwinkeltheit (log) : %lf\n",grenz50[8]);
        fprintf(stdprn,"Obere Grenze für Umfang/Fläche : %lf\n",grenz50[5]);

        fprintf(stdprn,"\nErkennung anderer Regionen:\n");
        fprintf(stdprn,"Obere Grenze für die Fläche (log) : %lf\n",grenz2[0]);
        fprintf(stdprn,"Obere Grenze für den Mittelwert : %lf\n",grenz2[1]);
        fprintf(stdprn,"Obere Grenze für das Maximum (log) : %lf\n",grenz2[2]);
        fprintf(stdprn,"Obere Grenze für den Umfang (log) : %lf\n",grenz2[4]);
        fprintf(stdprn,"Untere Grenze für die Rundheit (log) : %lf\n",grenz2[6]);
        fprintf(stdprn,"Obere Grenze für die Verwinkeltheit (log) : %lf\n",grenz2[8]);
        fprintf(stdprn,"Untere Grenze für Umfang/Fläche : %lf\n",grenz2[5]);

        fprintf(stdprn,"\n\nAnzahl der Extravasationsregionen: %d\n",anz[0]);
        fprintf(stdprn,"Anzahl der anderen Regionen      : %d\n",anz[1]);
        fprintf(stdprn,"\nErkannter Prozentsatz der Extravasationsregionen :
%3.2lf%%\n",p50[0]);
        fprintf(stdprn,"Dabei falsch erkannter Prozentsatz anderer Regionen :
%3.2lf%%\n",p50[1]);
        fprintf(stdprn,"Prozentsatz der Extravasationen von den Regionen,\ndie die Bedingung
erfüllen : %3.2lf%%\n",per50);
        fprintf(stdprn,"\nErkannter Prozentsatz der anderen Regionen : %3.2lf%%\n",p2[1]);
        fprintf(stdprn,"Dabei falsch erkannter Prozentsatz der Extravasationsregionen:
%3.2lf%%\n",p2[0]);
        fprintf(stdprn,"Prozentsatz der anderen Regionen von den Regionen,\ndie die
Bedingung erfüllen : %3.2lf%%\n",per2);
        }


}
```

7.14.UND-Verknüpfung

Die Beschreibungskriterien der Extravasationsregionen müssen spaltenweise in der Datei G:\DATEN\DATEN1.TXT abgespeichert sein. Über jeder Spalte sollte die Bezeichnung der Spalte stehen. Dasselbe gilt für die anderen Regionen. Hier heißt die Datei aber G:\DATEN\DATEN2.TXT.

Das folgende Programm trägt den Namen ZUS_UND.C.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

void main();

void main()
{
FILE *Datei;
char Dateiname[25]={"G:\\DATEN\\DATEN1.TXT"};
char Dateiname2[25]={"G:\\DATEN\\DATEN2.TXT"};
char dummy[15];
double daten[9],grenz50[9],grenz2[9],p50[2],p2[2],per2,per50;
int i,anz[2],a2[2],a50[2];
long curpos,length;
char k,key;


printf("\n\n\nUND-Verknüpfung\n================\n\n\n");

/*************************************************************************/
printf("\n\nDateneingabe:\n\n");
/*************************************************************************/

printf("\nExtravasationsregionenerkennung:\n\n");
printf("Untere Grenze für die Fläche (log) ? ");
scanf("%lf",&grenz50[0]);
if(grenz50[0]==0.0) grenz50[0]=3.15;
printf("Untere Grenze für das Maximum (log) ? ");
scanf("%lf",&grenz50[2]);
if(grenz50[2]==0.0) grenz50[2]=1.925;
printf("Untere Grenze für den Umfang (log) ? ");
scanf("%lf",&grenz50[4]);
if(grenz50[4]==0.0) grenz50[4]=2.8;
printf("Untere Grenze für die Verwinkeltheit (log) ? ");
scanf("%lf",&grenz50[8]);
if(grenz50[8]==0.0) grenz50[8]=2.44;
printf("Obere Grenze für Umfang/Fläche ? ");
scanf("%lf",&grenz50[5]);
if(grenz50[5]==0.0) grenz50[5]=0.2995;

printf("\n\nErkennung anderer Regionen:\n\n");
printf("Obere Grenze für die Fläche (log) ? ");
scanf("%lf",&grenz2[0]);
if(grenz2[0]==0.0) grenz2[0]=1.486895;
printf("Obere Grenze für den Mittelwert ? ");
scanf("%lf",&grenz2[1]);
```

```
if(grenz2[1]==0.0) grenz2[1]=12.613793;
printf("Obere Grenze für das Maximum (log) ? ");
scanf("%lf",&grenz2[2]);
if(grenz2[2]==0.0) grenz2[2]=1.220618;
printf("Obere Grenze für den Umfang (log) ? ");
scanf("%lf",&grenz2[4]);
if(grenz2[4]==0.0) grenz2[4]=1.430855;
printf("Untere Grenze für die Rundheit (log) ? ");
scanf("%lf",&grenz2[6]);
if(grenz2[6]==0.0) grenz2[6]=0.023674;
printf("Obere Grenze für die Verwinkeltheit (log) ? ");
scanf("%lf",&grenz2[8]);
if(grenz2[8]==0.0) grenz2[8]=0.911499;
printf("Untere Grenze für Umfang/Fläche ? ");
scanf("%lf",&grenz2[5]);
if(grenz2[5]==0.0) grenz2[5]=0.932414;
printf("\n\n");

/*************************************************************************/
/* Daten lesen */
/*************************************************************************/

for(k=0;k<2;k++)
        {
        if(k==1) strcpy(Dateiname,Dateiname2);
        printf("Daten %d lesen\n",k+1);
        if ((Datei=fopen(Dateiname,"rt"))==NULL)
                {
                printf("\a\aError: Kann die Datei '%s' nicht öffnen!\a\a\n",Dateiname);
                exit(0);
                }

        curpos=ftell(Datei);
        fseek(Datei, 0L, SEEK_END);
        length=ftell(Datei);
        fseek(Datei, curpos, SEEK_SET);

        for(i=1;i<=9;i++)
                fscanf(Datei,"%s",dummy);

        curpos=ftell(Datei);

        anz[k]=0;
        a50[k]=0;
        a2[k]=0;
        while(curpos<length)
                {
                for(i=0;i<9;i++)
                        fscanf(Datei,"%lf",&daten[i]);

                curpos=ftell(Datei);

                if(curpos<length)
                        {
                        daten[0]=log10(daten[0]);
                        daten[2]=log10(daten[2]);
                        daten[4]=log10(daten[4]);
                        daten[6]=log10(daten[6]);
                        daten[8]=log10(daten[8]);
```

```
                                anz[k]++;
                                if((daten[0]>grenz50[0])
                                 &&(daten[2]>grenz50[2])
                                 &&(daten[4]>grenz50[4])
                                 &&(daten[8]>grenz50[8])
                                 &&(daten[5]<=grenz50[5]))
                                        a50[k]++;

                                if((daten[0]<=grenz2[0])
                                 &&(daten[1]<=grenz2[1])
                                 &&(daten[2]<=grenz2[2])
                                 &&(daten[4]<=grenz2[4])
                                 &&(daten[6]>grenz2[6])
                                 &&(daten[8]<=grenz2[8])
                                 &&(daten[5]>grenz2[5]))
                                        a2[k]++;
                                }

                        }
                p50[k]=100.0*(double)a50[k]/(double)anz[k];
                p2[k]=100.0*(double)a2[k]/(double)anz[k];
                fclose(Datei);
                }
per2=100.0*(double)a2[1]/(double)(a2[0]+a2[1]);
per50=100.0*(double)a50[0]/(double)(a50[0]+a50[1]);

/*************************************************************************/
printf("\nErgebnisausgabe:\n");
/*************************************************************************/

printf("\nExtravasationsregionenerkennung:\n");
printf("Untere Grenze für die Fläche (log) : %lf\n",grenz50[0]);
printf("Untere Grenze für das Maximum (log) : %lf\n",grenz50[2]);
printf("Untere Grenze für den Umfang (log) : %lf\n",grenz50[4]);
printf("Untere Grenze für die Verwinkeltheit (log) : %lf\n",grenz50[8]);
printf("Obere Grenze für Umfang/Fläche : %lf\n",grenz50[5]);

printf("\nErkennung anderer Regionen:\n");
printf("Obere Grenze für die Fläche (log) : %lf\n",grenz2[0]);
printf("Obere Grenze für den Mittelwert : %lf\n",grenz2[1]);
printf("Obere Grenze für das Maximum (log) : %lf\n",grenz2[2]);
printf("Obere Grenze für den Umfang (log) : %lf\n",grenz2[4]);
printf("Untere Grenze für die Rundheit (log) : %lf\n",grenz2[6]);
printf("Obere Grenze für die Verwinkeltheit (log) : %lf\n",grenz2[8]);
printf("Untere Grenze für Umfang/Fläche : %lf\n",grenz2[5]);

printf("\n\nAnzahl der Extravasationsregionen: %d\n",anz[0]);
printf("Anzahl der anderen Regionen      : %d\n",anz[1]);
printf("\nErkannter Prozentsatz der Extravasationsregionen : %3.2lf%%\n",p50[0]);
printf("Dabei falsch erkannter Prozentsatz anderer Regionen : %3.2lf%%\n",p50[1]);
printf("Prozentsatz der Extravasationen von den Regionen,\ndie die Bedingung erfüllen :
%3.2lf%%\n",per50);
printf("\nErkannter Prozentsatz der anderen Regionen : %3.2lf%%\n",p2[1]);
printf("Dabei falsch erkannter Prozentsatz der Extravasationsregionen: %3.2lf%%\n",p2[0]);
printf("Prozentsatz der anderen Regionen von den Regionen,\ndie die Bedingung erfüllen :
%3.2lf%%\n",per2);

printf("\n\nDrucken [J/(N)] ? ");
```

```
key=toupper(getch());
printf("%c\n",key);
if(key=='J')
        {
        fprintf(stdprn,"\n\n\nUND-Verknüpfung\n===============\n\n\n");
        fprintf(stdprn,"Extravasationsregionenerkennung:\n");
        fprintf(stdprn,"Untere Grenze für die Fläche (log) : %lf\n",grenz50[0]);
        fprintf(stdprn,"Untere Grenze für das Maximum (log) : %lf\n",grenz50[2]);
        fprintf(stdprn,"Untere Grenze für den Umfang (log) : %lf\n",grenz50[4]);
        fprintf(stdprn,"Untere Grenze für die Verwinkeltheit (log) : %lf\n",grenz50[8]);
        fprintf(stdprn,"Obere Grenze für Umfang/Fläche : %lf\n",grenz50[5]);

        fprintf(stdprn,"\nErkennung anderer Regionen:\n");
        fprintf(stdprn,"Obere Grenze für die Fläche (log) : %lf\n",grenz2[0]);
        fprintf(stdprn,"Obere Grenze für den Mittelwert : %lf\n",grenz2[1]);
        fprintf(stdprn,"Obere Grenze für das Maximum (log) : %lf\n",grenz2[2]);
        fprintf(stdprn,"Obere Grenze für den Umfang (log) : %lf\n",grenz2[4]);
        fprintf(stdprn,"Untere Grenze für die Rundheit (log) : %lf\n",grenz2[6]);
        fprintf(stdprn,"Obere Grenze für die Verwinkeltheit (log) : %lf\n",grenz2[8]);
        fprintf(stdprn,"Untere Grenze für Umfang/Fläche : %lf\n",grenz2[5]);

        fprintf(stdprn,"\n\nAnzahl der Extravasationsregionen: %d\n",anz[0]);
        fprintf(stdprn,"Anzahl der anderen Regionen      : %d\n",anz[1]);
        fprintf(stdprn,"\nErkannter Prozentsatz der Extravasationsregionen :
%3.2lf%%\n",p50[0]);
        fprintf(stdprn,"Dabei falsch erkannter Prozentsatz anderer Regionen :
%3.2lf%%\n",p50[1]);
        fprintf(stdprn,"Prozentsatz der Extravasationen von den Regionen,\ndie die Bedingung
erfüllen : %3.2lf%%\n",per50);
        fprintf(stdprn,"\nErkannter Prozentsatz der anderen Regionen : %3.2lf%%\n",p2[1]);
        fprintf(stdprn,"Dabei falsch erkannter Prozentsatz der Extravasationsregionen:
%3.2lf%%\n",p2[0]);
        fprintf(stdprn,"Prozentsatz der anderen Regionen von den Regionen,\ndie die
Bedingung erfüllen : %3.2lf%%\n",per2);
        }
}
```

7.15.Ketten-Verknüpfung

Die Beschreibungskriterien der Extravasationsregionen müssen spaltenweise in der Datei G:\DATEN\DATEN1.TXT abgespeichert sein. Über jeder Spalte sollte die Bezeichnung der Spalte stehen. Dasselbe gilt für die anderen Regionen. Hier heißt die Datei aber G:\DATEN\DATEN2.TXT.

Das folgende Programm trägt den Namen ZUS_KET.C.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

void main();

void main()
{
FILE *Datei;
char Dateiname[25]={"G:\\DATEN\\DATEN1.TXT"};
char Dateiname2[25]={"G:\\DATEN\\DATEN2.TXT"};
char dummy[15],text0[15],text1[15];
double daten[9],grenz50[9],grenz2[9];
double p50[2],p2[2],ps50[2],ps2[2],psn50[2],psn2[2];
double per2,per50,pers2,pers50,persn2,persn50;
int i,anz[2],a2[2],a50[2],as2[2],as50[2],asn2[2],asn50[2],sp[2];
long curpos,length;
char k,key,b2[9],b50[9],flag;


printf("\n\n\nKetten-Verknüpfung\n==================\n\n\n");

/*************************************************************************/
printf("\n\nDateneingabe:\n\n");
/*************************************************************************/

printf("\nPrimäre Datenspalte ? ");
scanf("%d",&sp[0]);
printf("Sekundäre Datenspalte ? ");
scanf("%d",&sp[1]);

printf("\nExtravasationsregionenerkennung:\n\n");
if((sp[0]==0)||(sp[1]==0))
        {
        printf("Untere Grenze für die Fläche (log) ? ");
        scanf("%lf",&grenz50[0]);
        if(grenz50[0]==0.0) grenz50[0]=3.15;
        }

if((sp[0]==2)||(sp[1]==2))
        {
        printf("Untere Grenze für das Maximum (log) ? ");
        scanf("%lf",&grenz50[2]);
        if(grenz50[2]==0.0) grenz50[2]=1.925;
        }

if((sp[0]==4)||(sp[1]==4))
```

```
        {
        printf("Untere Grenze für den Umfang (log) ? ");
        scanf("%lf",&grenz50[4]);
        if(grenz50[4]==0.0) grenz50[4]=2.8;
        }

if((sp[0]==8)||(sp[1]==8))
        {
        printf("Untere Grenze für die Verwinkeltheit (log) ? ");
        scanf("%lf",&grenz50[8]);
        if(grenz50[8]==0.0) grenz50[8]=2.44;
        }

if((sp[0]==5)||(sp[1]==5))
        {
        printf("Obere Grenze für Umfang/Fläche ? ");
        scanf("%lf",&grenz50[5]);
        if(grenz50[5]==0.0) grenz50[5]=0.2995;
        }

printf("\n\nErkennung anderer Regionen:\n\n");
if((sp[0]==0)||(sp[1]==0))
        {
        printf("Obere Grenze für die Fläche (log) ? ");
        scanf("%lf",&grenz2[0]);
        if(grenz2[0]==0.0) grenz2[0]=1.486895;
        }

if((sp[0]==1)||(sp[1]==1))
        {
        printf("Obere Grenze für den Mittelwert ? ");
        scanf("%lf",&grenz2[1]);
        if(grenz2[1]==0.0) grenz2[1]=12.613793;
        }

if((sp[0]==2)||(sp[1]==2))
        {
        printf("Obere Grenze für das Maximum (log) ? ");
        scanf("%lf",&grenz2[2]);
        if(grenz2[2]==0.0) grenz2[2]=1.220618;
        }

if((sp[0]==4)||(sp[1]==4))
        {
        printf("Obere Grenze für den Umfang (log) ? ");
        scanf("%lf",&grenz2[4]);
        if(grenz2[4]==0.0) grenz2[4]=1.430855;
        }

if((sp[0]==6)||(sp[1]==6))
        {
        printf("Untere Grenze für die Rundheit (log) ? ");
        scanf("%lf",&grenz2[6]);
        if(grenz2[6]==0.0) grenz2[6]=0.023674;
        }

if((sp[0]==8)||(sp[1]==8))
        {
        printf("Obere Grenze für die Verwinkeltheit (log) ? ");
```

```
        scanf("%lf",&grenz2[8]);
        if(grenz2[8]==0.0) grenz2[8]=0.911499;
        }

if((sp[0]==5)||(sp[1]==5))
        {
        printf("Untere Grenze für Umfang/Fläche ? ");
        scanf("%lf",&grenz2[5]);
        if(grenz2[5]==0.0) grenz2[5]=0.932414;
        }

printf("\n\n");

/*************************************************************************/
/* Daten lesen */
/*************************************************************************/

for(k=0;k<2;k++)
        {
        if(k==1)
                strcpy(Dateiname,Dateiname2);
        printf("Daten %d lesen\n",k+1);

        if ((Datei=fopen(Dateiname,"rt"))==NULL)
                {
                printf("\a\aError: Kann die Datei '%s' nicht öffnen!\a\a\n",Dateiname);
                exit(0);
                }

        curpos=ftell(Datei);
        fseek(Datei, 0L, SEEK_END);
        length=ftell(Datei);
        fseek(Datei, curpos, SEEK_SET);

        for(i=0;i<9;i++)
                {
                fscanf(Datei,"%s",dummy);
                if(i==sp[0])
                        strcpy(text0,dummy);
                if(i==sp[1])
                        strcpy(text1,dummy);
                }

        curpos=ftell(Datei);

        anz[k]=0;
        a50[k]=0;
        a2[k]=0;
        as50[k]=0;
        as2[k]=0;
        asn50[k]=0;
        asn2[k]=0;
        while(curpos<length)
                {
                for(i=0;i<9;i++)
                        fscanf(Datei,"%lf",&daten[i]);

                curpos=ftell(Datei);
```

```
if(curpos<length)
        {
        daten[0]=log10(daten[0]);
        daten[2]=log10(daten[2]);
        daten[4]=log10(daten[4]);
        daten[6]=log10(daten[6]);
        daten[7]=log10(daten[7]);
        daten[8]=log10(daten[8]);

        anz[k]++;
        for(i=0;i<9;i++)
                {
                b2[i]=0;
                b50[i]=0;
                }

        if(daten[0]>grenz50[0])
                b50[0]=1;
        if(daten[2]>grenz50[2])
                b50[2]=1;
        if(daten[4]>grenz50[4])
                b50[4]=1;
        if(daten[8]>grenz50[8])
                b50[8]=1;
        if(daten[5]<=grenz50[5])
                b50[5]=1;

        if(daten[0]<=grenz2[0])
                b2[0]=1;
        if(daten[1]<=grenz2[1])
                b2[1]=1;
        if(daten[2]<=grenz2[2])
                b2[2]=1;
        if(daten[4]<=grenz2[4])
                b2[4]=1;
        if(daten[6]>grenz2[6])
                b2[6]=1;
        if(daten[8]<=grenz2[8])
                b2[8]=1;
        if(daten[5]>grenz2[5])
                b2[5]=1;

        flag=0;
        if(b50[sp[0]])
                {
                a50[k]++;
                flag=1;
                }
        if(b50[sp[1]]&&!flag)
                as50[k]++;
        if(b50[sp[1]]&&flag)
                asn50[k]++;

        flag=0;
        if(b2[sp[0]])
                {
                a2[k]++;
                flag=1;
                }
```

```
                        if(b2[sp[1]]&&!flag)
                                as2[k]++;
                        if(b2[sp[1]]&&flag)
                                asn2[k]++;

                        }

                }
        p50[k]=100.0*(double)a50[k]/(double)anz[k];
        p2[k]=100.0*(double)a2[k]/(double)anz[k];
        ps50[k]=100.0*(double)as50[k]/(double)anz[k];
        ps2[k]=100.0*(double)as2[k]/(double)anz[k];
        psn50[k]=100.0*(double)asn50[k]/(double)anz[k];
        psn2[k]=100.0*(double)asn2[k]/(double)anz[k];
        fclose(Datei);
        }
per2=100.0*(double)a2[1]/(double)(a2[0]+a2[1]);
per50=100.0*(double)a50[0]/(double)(a50[0]+a50[1]);
pers2=100.0*(double)as2[1]/(double)(as2[0]+as2[1]);
pers50=100.0*(double)as50[0]/(double)(as50[0]+as50[1]);
persn2=100.0*(double)asn2[1]/(double)(asn2[0]+asn2[1]);
persn50=100.0*(double)asn50[0]/(double)(asn50[0]+asn50[1]);

/*************************************************************************/
printf("\nErgebnisausgabe:\n");
/*************************************************************************/

printf("\nExtravasationsregionenerkennung:\n");
if((sp[0]==0)||(sp[1]==0))
        printf("Untere Grenze für die Fläche (log) : %lf\n",grenz50[0]);
if((sp[0]==2)||(sp[1]==2))
        printf("Untere Grenze für das Maximum (log) : %lf\n",grenz50[2]);
if((sp[0]==4)||(sp[1]==4))
        printf("Untere Grenze für den Umfang (log) : %lf\n",grenz50[4]);
if((sp[0]==8)||(sp[1]==8))
        printf("Untere Grenze für die Verwinkeltheit (log) : %lf\n",grenz50[8]);
if((sp[0]==5)||(sp[1]==5))
        printf("Obere Grenze für Umfang/Fläche : %lf\n",grenz50[5]);

printf("\nErkennung anderer Regionen:\n");
if((sp[0]==0)||(sp[1]==0))
        printf("Obere Grenze für die Fläche (log) : %lf\n",grenz2[0]);
if((sp[0]==1)||(sp[1]==1))
        printf("Obere Grenze für den Mittelwert : %lf\n",grenz2[1]);
if((sp[0]==2)||(sp[1]==2))
        printf("Obere Grenze für das Maximum (log) : %lf\n",grenz2[2]);
if((sp[0]==4)||(sp[1]==4))
        printf("Obere Grenze für den Umfang (log) : %lf\n",grenz2[4]);
if((sp[0]==6)||(sp[1]==6))
        printf("Untere Grenze für die Rundheit (log) : %lf\n",grenz2[6]);
if((sp[0]==8)||(sp[1]==8))
        printf("Obere Grenze für die Verwinkeltheit (log) : %lf\n",grenz2[8]);
if((sp[0]==5)||(sp[1]==5))
        printf("Untere Grenze für Umfang/Fläche : %lf\n",grenz2[5]);

printf("\n\nAnzahl der Extravasationsregionen: %d\n",anz[0]);
printf("Anzahl der anderen Regionen      : %d\n",anz[1]);
printf("Primärdaten                      : %s\n",text0);
printf("Sekundärdaten                    : %s\n",text1);
```

```
printf("\nErkannter Prozentsatz der Extravasationsregionen mit Primärdaten:
%3.2lf%%\n",p50[0]);
printf("Zusätzlich erkannte Extravasationsregionen mit Sekundärdaten: %3.2lf%%\n",ps50[0]);
printf("Bereits erkannte Extravasationsregionen mit Sekundärdaten: %3.2lf%%\n",psn50[0]);
printf("Dabei falsch erkannter Prozentsatz anderer Regionen mit Primärdaten:
%3.2lf%%\n",p50[1]);
printf("Zusätzlich falsch erkannte andere Regionen mit Sekundärdaten: %3.2lf%%\n",ps50[1]);
printf("Bereits falsch erkannte andere Regionen mit Sekundärdaten: %3.2lf%%\n",psn50[1]);
printf("Prozentsatz der Extravasationen von allen Regionen,\ndie die Bedingung der Primärdaten
erfüllen: %3.2lf%%\n",per50);
printf("Prozentsatz der Extravasationen von allen Regionen,\ndie zusätzlich die Bedingung der
Sekundärdaten erfüllen: %3.2lf%%\n",pers50);
printf("Prozentsatz der Extravasationen von allen Regionen,\ndie bereits die Bedingung der
Sekundärdaten erfüllten: %3.2lf%%\n",persn50);

printf("\nErkannter Prozentsatz der anderen Regionen mit Primärdaten: %3.2lf%%\n",p2[1]);
printf("Zusätzlich erkannte andere Regionen mit Sekundärdaten: %3.2lf%%\n",ps2[1]);
printf("Bereits erkannte andere Regionen mit Sekundärdaten: %3.2lf%%\n",psn2[1]);
printf("Dabei falsch erkannte Extravasationsregionen mit Primärdaten: %3.2lf%%\n",p2[0]);
printf("Zusätzlich falsch erkannte Extravasationsregionen mit Sekundärdaten:
%3.2lf%%\n",ps2[0]);
printf("Bereits falsch erkannte Extravasationsregionen mit Sekundärdaten:
%3.2lf%%\n",psn2[0]);
printf("Prozentsatz der anderen Regionen von allen Regionen,\ndie die Bedingung der
Primärdaten erfüllen: %3.2lf%%\n",per2);
printf("Prozentsatz der anderen Regionen von allen Regionen,\ndie zusätzlich die Bedingung der
Sekundärdaten erfüllen: %3.2lf%%\n",pers2);
printf("Prozentsatz der anderen Regionen von allen Regionen,\ndie bereits die Bedingung der
Sekundärdaten erfüllten: %3.2lf%%\n",persn2);

printf("\n\nDrucken [J/(N)] ? ");
key=toupper(getch());
printf("%c\n",key);
if(key=='J')
        {
        fprintf(stdprn,"\nKetten-Verknüpfung\n==================\n\n\n");
        fprintf(stdprn,"Extravasationsregionenerkennung:\n");
        if((sp[0]==0)||(sp[1]==0))
                fprintf(stdprn,"Untere Grenze für die Fläche (log) : %lf\n",grenz50[0]);
        if((sp[0]==2)||(sp[1]==2))
                fprintf(stdprn,"Untere Grenze für das Maximum (log) : %lf\n",grenz50[2]);
        if((sp[0]==4)||(sp[1]==4))
                fprintf(stdprn,"Untere Grenze für den Umfang (log) : %lf\n",grenz50[4]);
        if((sp[0]==8)||(sp[1]==8))
                fprintf(stdprn,"Untere Grenze für die Verwinkeltheit (log) : %lf\n",grenz50[8]);
        if((sp[0]==5)||(sp[1]==5))
                fprintf(stdprn,"Obere Grenze für Umfang/Fläche : %lf\n",grenz50[5]);

        fprintf(stdprn,"\nErkennung anderer Regionen:\n");
        if((sp[0]==0)||(sp[1]==0))
                fprintf(stdprn,"Obere Grenze für die Fläche (log) : %lf\n",grenz2[0]);
        if((sp[0]==1)||(sp[1]==1))
                fprintf(stdprn,"Obere Grenze für den Mittelwert : %lf\n",grenz2[1]);
        if((sp[0]==2)||(sp[1]==2))
                fprintf(stdprn,"Obere Grenze für das Maximum (log) : %lf\n",grenz2[2]);
        if((sp[0]==4)||(sp[1]==4))
                fprintf(stdprn,"Obere Grenze für den Umfang (log) : %lf\n",grenz2[4]);
        if((sp[0]==6)||(sp[1]==6))
```

```
                fprintf(stdprn,"Untere Grenze für die Rundheit (log) : %lf\n",grenz2[6]);
        if((sp[0]==8)||(sp[1]==8))
                fprintf(stdprn,"Obere Grenze für die Verwinkeltheit (log) : %lf\n",grenz2[8]);
        if((sp[0]==5)||(sp[1]==5))
                fprintf(stdprn,"Untere Grenze für Umfang/Fläche : %lf\n",grenz2[5]);

        fprintf(stdprn,"\n\nAnzahl der Extravasationsregionen: %d\n",anz[0]);
        fprintf(stdprn,"Anzahl der anderen Regionen      : %d\n",anz[1]);
        fprintf(stdprn,"Primärdaten                      : %s\n",text0);
        fprintf(stdprn,"Sekundärdaten                    : %s\n",text1);

        fprintf(stdprn,"\nErkannter Prozentsatz der Extravasationsregionen\n  mit Primärdaten:
%3.2lf%%\n",p50[0]);
        fprintf(stdprn,"Zusätzlich erkannte Extravasationsregionen mit Sekundärdaten:
%3.2lf%%\n",ps50[0]);
        fprintf(stdprn,"Bereits erkannte Extravasationsregionen mit Sekundärdaten:
%3.2lf%%\n",psn50[0]);
        fprintf(stdprn,"Dabei falsch erkannte andere Regionen mit Primärdaten:
%3.2lf%%\n",p50[1]);
        fprintf(stdprn,"Zusätzlich falsch erkannte andere Regionen mit Sekundärdaten:
%3.2lf%%\n",ps50[1]);
        fprintf(stdprn,"Bereits falsch erkannte andere Regionen mit Sekundärdaten:
%3.2lf%%\n",psn50[1]);
        fprintf(stdprn,"\nProzentsatz der Extravasationen von allen Regionen,\n  die die
Bedingung der Primärdaten erfüllen: %3.2lf%%\n",per50);
        fprintf(stdprn,"Prozentsatz der Extravasationen von allen Regionen,\n  die zusätzlich die
Bedingung der Sekundärdaten erfüllen: %3.2lf%%\n",pers50);
        fprintf(stdprn,"Prozentsatz der Extravasationen von allen Regionen,\n  die bereits die
Bedingung der Sekundärdaten erfüllten: %3.2lf%%\n",persn50);

        fprintf(stdprn,"\n\nErkannter Prozentsatz der anderen Regionen mit Primärdaten:
%3.2lf%%\n",p2[1]);
        fprintf(stdprn,"Zusätzlich erkannte andere Regionen mit Sekundärdaten:
%3.2lf%%\n",ps2[1]);
        fprintf(stdprn,"Bereits erkannte andere Regionen mit Sekundärdaten:
%3.2lf%%\n",psn2[1]);
        fprintf(stdprn,"Dabei falsch erkannte Extravasationsregionen mit Primärdaten:
%3.2lf%%\n",p2[0]);
        fprintf(stdprn,"Zusätzlich falsch erkannte Extravasationsregionen\n  mit Sekundärdaten:
%3.2lf%%\n",ps2[0]);
        fprintf(stdprn,"Bereits falsch erkannte Extravasationsregionen mit\n  Sekundärdaten:
%3.2lf%%\n",psn2[0]);
        fprintf(stdprn,"\nProzentsatz der anderen Regionen von allen Regionen,\n  die die
Bedingung der Primärdaten erfüllen: %3.2lf%%\n",per2);
        fprintf(stdprn,"Prozentsatz der anderen Regionen von allen Regionen,\n  die zusätzlich
die Bedingung der Sekundärdaten erfüllen: %3.2lf%%\n",pers2);
        fprintf(stdprn,"Prozentsatz der anderen Regionen von allen Regionen,\n  die bereits die
Bedingung der Sekundärdaten erfüllten: %3.2lf%%\n",persn2);
        }
}
```

8.Literaturverzeichnis

[1] Lenz, Reimar: Ein Verfahren zur Schätzung der Parameter Geometrischer Bildtransformationen; Nachrichtentechnische Berichte Band 15; Schriftenreihe des Lehrstuhls für Nachrichtentechnik der Technischen Universität München; Hrsg.: H. Marko und G. Binkert; München 1986.

[2] Liedtke, Claus-Eberhard/ Ender, Manfred: Wissensbasierte Bildverarbeitung; Nachrichtentechnik Band 19; Hrsg.: H. Marko; Springer-Verlag Berlin Heidelberg New York London Paris Tokyo Hong Kong 1989.

[3] Wahl, Friedrich M.: Digitale Bildsignalverarbeitung; Nachrichtentechnik Band 13; Hrsg.: H. Marko; Springer-Verlag Berlin Heidelberg New York Tokyo 1984.

[4] Haberäcker, Peter: Digitale Bildverarbeitung; Hanser Studienbücher; 3. Auflage; Carl Hanser Verlag München Wien1989.

[5] Niemann, Heinrich: Klassifikation von Mustern; Springer-Verlag Berlin Heidelberg New York Tokyo 1983.

[6]Gonzalez, Rafael C./ Wintz, Paul: Digital Image Processing; Second Edition; Addison-Wesley Publishing Company 1987.