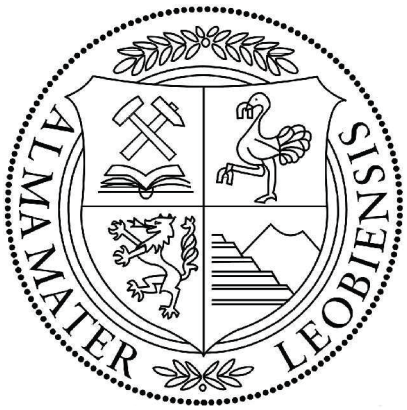
Montanuniversität Leoben
Studienrichtung Industrieller Umweltschutz



DIPLOMARBEIT

am

Institut für Automation

zum Thema

Bildverarbeitungs- und geometrische Methoden zur Vermessung von 3D-Objekten

von

Georg Ernst Michael EGGER

An dieser Stelle erkläre ich Eides statt, die vorliegende Diplomarbeit selbstständig und nur unter Verwendung der angegebenen Literatur erstellt zu haben.

Leoben, 10. Februar 2002 (Egger Georg)

Mein Dank gilt

Herrn O.Univ.Prof.Dipl.-Ing.Dr.techn.

Paul O‘LEARY

für die Betreuung und Unterstützung bei all meinen Anliegen, sowie
für das von ihm geprägte gute Arbeitsklima dieses Institutes.

Dem Unternehmen

BÖHLER-EDELSTAHL

danke ich für seine Unterstützung, die sehr zum Gelingen
dieses Projekts beigetragen hat.

An dieser Stelle möchte ich mich auch bei dem Herrn

Dipl.-Ing. Mark Tratnig

bedanken, da wir uns bei Problemen jeder Art immer gegenseitig zu helfen wußten.

Nicht zuletzt bedanke ich mich an dieser Stelle bei meinen **Eltern**, die
mir das Studium an der Montanuniversität Leoben ermöglicht haben.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Aufgabenstellung

Im Rahmen der Diplomarbeit am Institut für Automation der Montanuniversität Leoben werden Möglichkeiten gesucht, ein bestimmtes 3D-Objekt, das sich durch einen 3D-Raum bewegt, mit mehreren Kameras zu erkennen, zu verfolgen und auch dessen Geometrie zu bestimmen.

Es geht hier grundsätzlich um die Möglichkeit Fertigungsprozesse zu automatisieren, in denen eine Objektverfolgung von wesentlicher Bedeutung ist.

Die Werkzeugfindung wurde deshalb so allgemein wie möglich gehalten um das Spektrum der Anwendungsmöglichkeiten nicht zu beschränken.

Grundsätzlich geht es in dieser Diplomarbeit darum, zu ermitteln, ob das Ziel der Objektverfolgung überhaupt zu erreichen ist und dann in weiterer Folge eine Lösungsmöglichkeit zu präsentieren.

Dieses Projekt wurde in Zusammenarbeit mit dem Konzern **Böhler-Uddeholm** entwickelt, der das Konzept auch unterstützt hat. Da es sich bei den zu verfolgenden Objekten um große Stahlblöcke handelt und deren Bewegung sehr genau beobachtet und gesteuert werden muss, wird auch eine große Anforderung an die Genauigkeit der Geometriebestimmung gelegt.

Das so gefundene Konzept stellt eine Möglichkeit dar die Geometrie und den Aufenthaltsort des Objektes zu bestimmen, ist aber aufgrund des Rechenaufwandes noch nicht für ein Echtzeitsystem geeignet.

1.1.1 Projektetappen:

1. Erfassung des Prinzips.
2. Durchführung einer Literaturrecherche zum Thema 3D.
3. Erarbeitung von projektiver Geometrie.
4. Auswahl eines geeigneten Konzeptes.
5. Verifizierung des Konzeptes mit einem Computerprogramm.
6. Überprüfung des Computerprogramms an Testbildern.

1.1.2 Didaktische Ziele:

1. Erwerb der Fähigkeit, ein Projekt selbstständig durchzuführen.
2. Erarbeitung von projektiver Geometrie.
3. Die für die Bildverarbeitung wesentlichen Programmiersprachen zu vertiefen und anzuwenden.
4. Messergebnisse und Testergebnisse kritisch beurteilen und entsprechend zu reagieren.
5. Wesentliche Aussagen zu einem Bericht zusammenzufassen um eine klare Dokumentation zu erreichen.

1.2 Stand der Technik

Visuelles Aufspüren bekannter, fester Gegenstände wie sie sich im Raum bewegen ist umfangreich studiert worden, da es eine entscheidende Vorbedingung für das Lösen von Aufgaben im Bereich der Robotertechnologie, biomedizinischen Bildanalyse, autonomer Navigation und Kontrolle ist.

Um Teilaufgaben zu lösen oder ihre Komplexität zu reduzieren, werden bestimmte Annahmen und Zwänge verhängt, um die Form und Bewegung der Gegenstände wie auch die Form der Umgebung zu beschreiben. In letzter Zeit wurde der Verarbeitung einer bestimmten Funktionalität mit höchstmöglicher Effizienz in Verbindung mit Beobachtung und Steuerung von Objekten eine noch höherere Bedeutung zugeordnet.

In allen diesen Versuchen ist die Computereffizienz immer noch einer der kritischsten Punkte, um den Erfolg der Lösung eines bestimmten Problems mit einem Kamerasystem zu gewährleisten.

Der Lösungsweg kann aber von zwei wesentlichen Entscheidungsmöglichkeiten beeinflusst werden. Einerseits sind das einmal die verschiedenen Kamerasystemkonzepte und andererseits die Bildverarbeitung selbst.

1.2.1 Kamerasysteme

1. Kombination von hoch- und niedrigauflösenden Kameras.
   Hier wird ein Kameraset verwendet, das aus Weitwinkelkameras mit niedriger Auflösung und aus beweglichen Kameras mit hoher Auflösung besteht, die dann das Objekt verfolgen und bemüht sind es immer im Zentrum des Bildes zu halten, damit Verzerrungen an den Rändern nicht so stark eingehen.

2. Kombination von 360 ° Panorama Kameras und CCD Kameras.
   In diesem Fall verwendet man zum Aufspüren der Objekte eine 360 ° Panorama Kamera, die extrem verzerrte Bilder liefert und dann wieder zur genaueren Bestimmung CCD Kameras.

3. Denselben Kameratyp an allen wichtigen Standorten.
   Dieses System wurde von uns gewählt, da man aufgrund des einheitlichen Kamerasystems auch eine einheitliche Softwarelösung verwenden kann.

1.2.2 Bildverarbeitung

Es wird kurz der Stand der Technik in der Bildverarbeitung in den Bereichen, die **für uns** von Interesse sind, erläutert.

1. Beobachtung des Objektes an einem gewissen Standpunkt.
   - Multi view (Stereoskopie).
     Hier wird jeder Standort mit mehreren Kameras überwacht und mit Hilfe von Stereoskopie das Objekt modelliert.
   - Monocular vision.
     Hier wird jeder Standort nur mit einer Kamera überwacht und mit Hilfe von gewissen gegebenen Informationen, z.B.: bekannte Ebenen im Raum, die mit dem Objekt zusammenfallen, Referenzpunkten usw., das Objekt modelliert.
2. Objektverfolgung.
   Bei der Objektverfolgung tritt das Stichwort blob auf. Ein blob ist ein Bereich gleicher Bildeigenschaften, die das zu verfolgende Objekt beschreiben. Das heißt, man legt fest, welche Bildeigenschaften hat das Objekt und bezieht nun seine Suche auf diese Bildeigenschaften. Hat man das Objekt nun gefunden, kann man seine Bildeigenschaften, mit denen man es extrahiert hat, nochmals aus den gefundenen Daten bestimmen und nun alle nachgeschalteten Operationen wie Konturfindung, Linienanpassung, Kurvenanpassung usw. durchführen.
3. Geometriebestimmung.
   Hier ist es jetzt möglich das 3D-Objekt wie schon oben beschrieben mit Stereoskopie zu bestimmen, d.h. das Objekt wird mit 2 oder mehreren Kameras aus möglichst verschiedenen Richtungen beobachtet und durch gemeinsame Bezugspunkte rekonstruiert. Dieses System ist unerlässlich bei Objektverfolgung von komplexen deformierbaren Körpern (z.B.: Finger, Mensch).
   Bei der Geometriebestimmung einfacher Objekte sollten aber vorerst Bilder einer Kamera und bekannte Bezugspunkte bzw. Ebenen im Raum als Datenmenge reichen.
4. Verwendetes Bildformat TIFF.

   Anwendungsbereich:

   - TIFF beschreibt Bilddaten die typisch von Druckern, frame grabbern, Zeichenprogrammen und Fotobearbeitungsprogrammen geliefert werden.
   - TIFF ist keine Druckersprache. Das Ziel der TIFF Dateien ist Raster-Bilddaten zu speichern und zu beschreiben.
   - Ein Hauptziel von TIFF Dateien ist eine Fülle von Möglichkeiten zur Verfügung zu stellen, in der Anwendungen Bilddaten austauschen können. Diese Fülle wird erfordert um die wechselnden Fähigkeiten von Scannern und anderen abbildenden Vorrichtungen auszunutzen.

- Obwohl TIFF ein umfangreiches Dateiformat ist, kann es für einfache Scanner und Anwendungen auch leicht benutzt werden, weil die Zahl der erforderten Felder klein ist.
- TIFF wird in gleichem Maße auf einer weiterführenden Basis gesteigert werden, wie neue Abbildungsbedürfnisse entstehen. Eine hohe Priorität ist strukturellen-TIFF Dateien zugeordnet, sodass zukünftige Steigerungen ohne großen Aufwand für Entwickler hinzugefügt werden können.
- TIFF kann Daten von Schwarzweiß-, Graustufen- und Farbbildern beschreiben.
- TIFF schliesst eine Zahl von Kompressionsmöglichkeiten ein, die Entwicklern erlauben, den besten Kompromiss zwischen Speicherplatz- und Zeitverbrauch für ihre Anwendungen zu wählen.
- TIFF wird zu Scannern, Druckern, oder Computeranzeigen nicht spezifisch gebunden.
- TIFF ist flexibel. Besondere Betriebssysteme, Dateisysteme, Kompilatoren oder Prozessoren werden nicht begünstigt.
- TIFF wurde entworfen um sich auch weiterzuentwickeln, abhängig von den Erfordernissen die entstehen.
- TIFF gestattet die Einschliessung einer unbeschränkten Anzahl von Privat- oder Spezialzweckinformationen.

Aufbau:

TIFF ist ein Bilddateiformat. Eine Datei ist definiert als eine Sequenz von 8-bit Bytes, wobei die Bytes von 0 bis N numeriert sind. Eine TIFF Datei beginnt mit einem **image file header**, also einer Art Zeiger, der auf ein Bilddateiverzeichnis (**image file directory** IFD) zeigt. Dieses IFD beinhaltet Informationen über das Bild sowie Zeiger, die auf die aktuellen Bilddaten zeigen.

Image File Header:

Eine TIFF Datei beginnt mit einem 8 Byte image file header, der folgende Informationen beinhaltet:

Bytes 0-1: Die Reihenfolge der Bytes, die innerhalb der Datei gebraucht wird. Zulässige Werte sind: II (4949.HEX) MM (4D4D.HEX). Im II Format ist die Reihenfolge der Bytes immer von dem niederwertigen Byte zu dem hochwertigen Byte, gültig für 16-bit und 32-bit integers. Das wird als little-endian Byte Reihenfolge bezeichnet. Im MM Format ist die Reihenfolge der Bytes immer von dem hochwertigen Byte zu dem niederwertigen Byte, gültig für 16-bit und 32-bit integers. Das wird als big-endian Byte Reihenfolge bezeichnet.

Bytes 2-3: Eine willkürlich aber sorgfältig gewählte Nummer (42) identifiziert die Datei als TIFF Datei. Die Reihenfolge der Bytes ist abhängig von den Bytes 0-1.

Bytes 4-7: Der Offset (in Bytes) des ersten IFD. Das Verzeichnis darf an jeder Stelle in dem File nach dem Header liegen, muss aber an einer Wort-Grenze (word boundary) beginnen.

Image File Directory IFD)

Ein IFD besteht aus einem 2-Byte Zähler, der die Anzahl der Verzeichniseinträge wiedergibt, gefolgt von einer Sequenz von 12 Byte Feldeinträgen, gefolgt von einem 4 Byte Offset des nächsten IFD (oder 0 bei keinem folgendem IFD). Es muss mindestens 1 IFD in einer TIFF Datei sein und jedes IFD muss mindestens einen Eintrag haben.

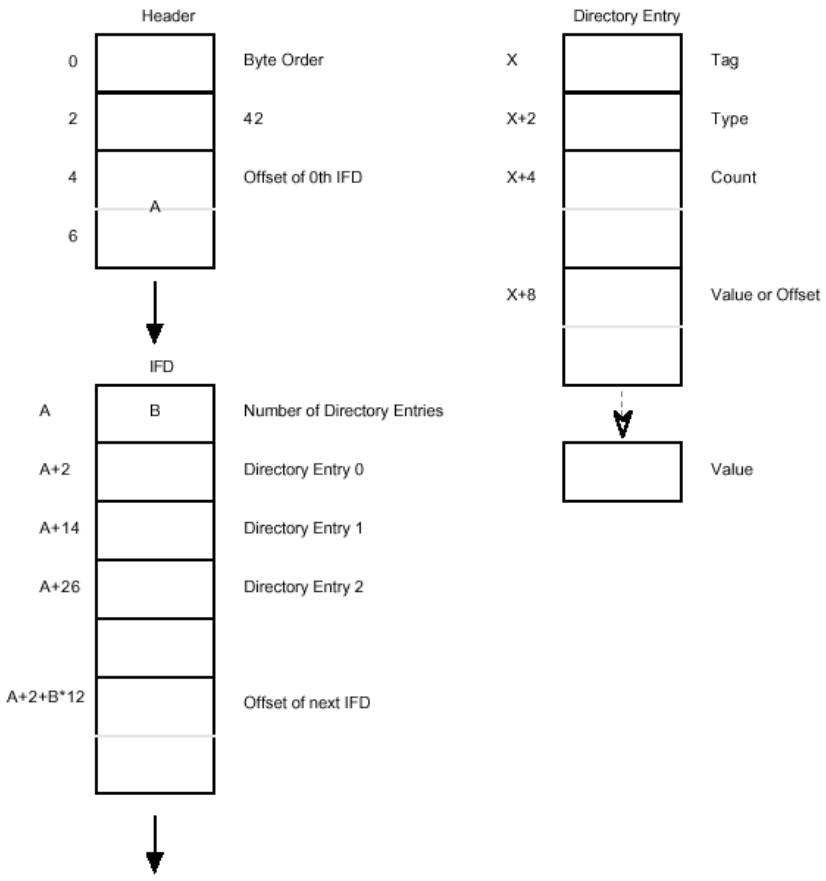
IFD Eintrag Jeder 12-Byte IFD Eintrag hat folgendes Format:

Bytes 0-1: Die Markierung, die das Feld identifiziert.

Bytes 2-3: Der Feldtyp.

Bytes 4-7: Die Anzahl der Werte, des angezeigten Typs.

Bytes 8-11 Der Wert-Offset, der Datei-Offset (in Bytes) des Wertes des Feldes. Es wird erwartet, dass der Wert an einer Wort-Grenze (word boundary) beginnt. Der dazugehörige Wert-Offset wird somit eine gerade Zahl sein.



1.3 Vorgehensweise

Das Ziel liegt darin aus 2D Bildern Informationen zu extrahieren, die genug Inhalt liefern um ein Abbild des Objektes in Form von Punkten, Linienzügen, Kreisen oder Kurven zu erschaffen, mit dem man im Endeffekt die Geometrie bestimmen kann.

Um jetzt einen Anhaltspunkt zu bekommen, welche Bilder man als Grundlage für das Rechenmodell erhält und mit welchen Problemen man schon am Anfang umgehen muss, wurden Bildaufnahmen vor Ort durchgeführt. Es handelt sich bei dem zu automatisierenden Vorgang um eine Walzlinie, in der heiße Stahlblöcke mit einem Förderband einem Walzwerk (Danieli Gerüst) zugeführt werden, wobei die Führung der Blöcke unter die Walze von vom Menschen gesteuerten Linealen durchgeführt wird.

1.3.1 Experimenteller Aufbau

- Messkette

Beschreibung	**Nummer**	**Typ**
Flächenkamera	KNr.1	Pulnix TM 6 CN
Objektive	ONr.1 ONr.2	6 mm 12.5mm
Visualisierungsrechner	VNR1	Industrie PC DEWE TRON 2000
Software Version		NT 4.0, Lab VIEW 4.0

Tabelle 1.1: Messkette

- Messpunkte in der Halle: Es wurden Messungen an folgenden 3 Punkten der Halle durchgeführt:
  - Danieli Gerüst
  - Einlauf Danieli Gerüst
  - Drehteller

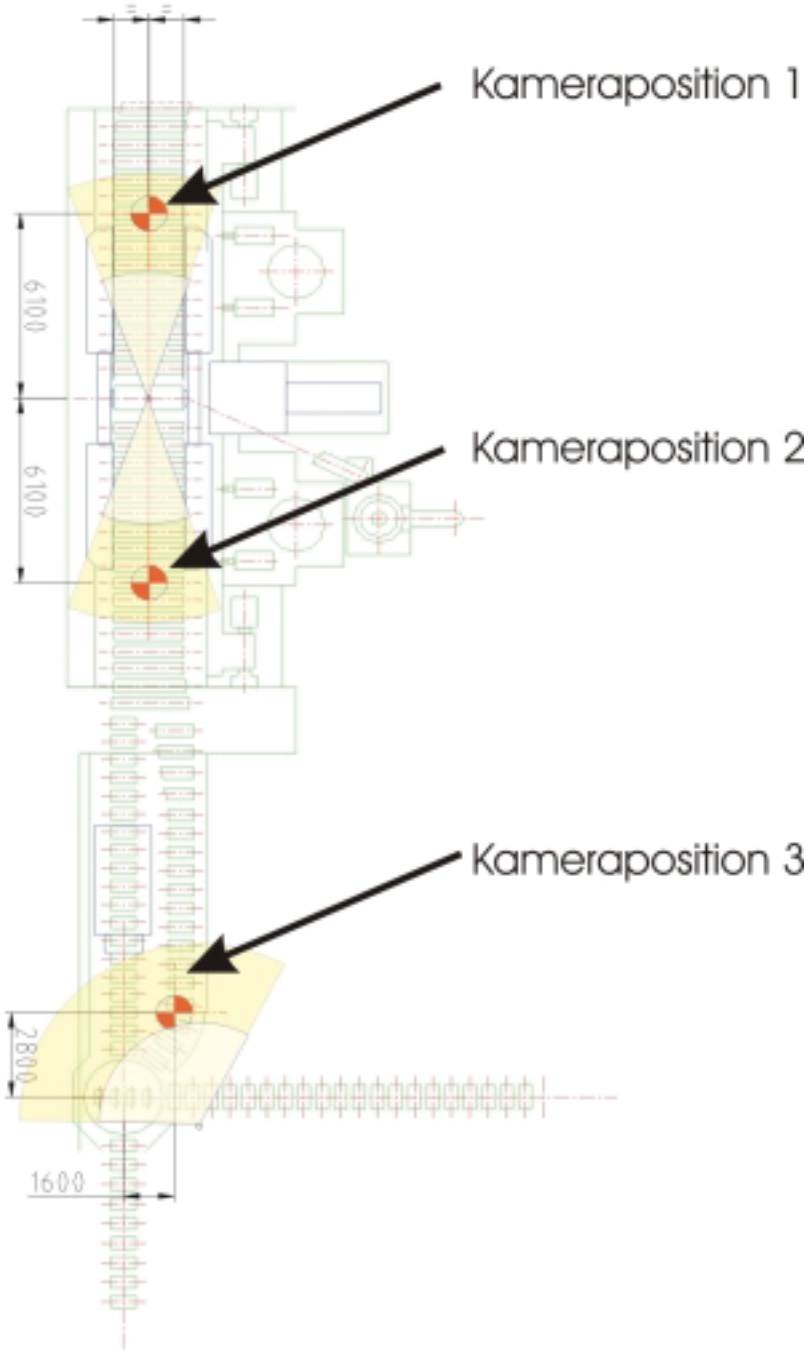


Abbildung 1.1: Kamerapositionen an Schlüsselstellen

- Kamerabefestigung
  Um das Objekt aus möglichst vielen verschiedenen Winkeln und Positionen betrachten zu können, wurde die Kamera auf einer bewegliche Halterung und auf einem drehbaren und ausführbaren Kragträger montiert.

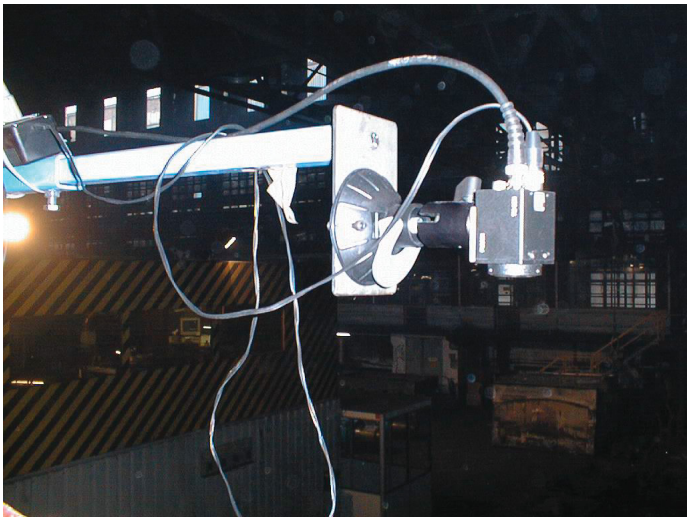


Abbildung 1.2: Bewegliche Kamerahalterung

Abbildung 1.3: Drehbarer ausfahrbarer Kragträger

- Erste Bilder
  Mit den richtigen Filtern und optimaler Gain-Einstellung kommt man zu einigermaßen brauchbaren Bildern. Das wird am besten mit einem Probebild verdeutlicht:

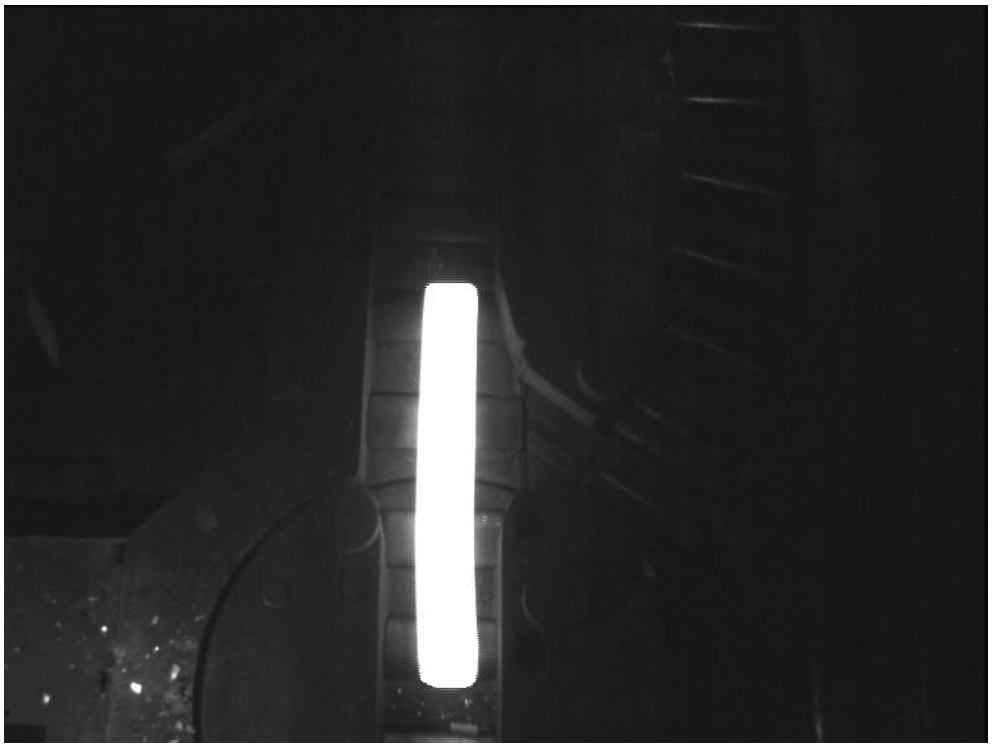


Abbildung 1.4: Stahlblock am Drehteller

1.3.2 Strategie der Erstellung des Rechenmodells

Benötigte Informationen

- Welche Information benötigt man, um das Objekt zu erfassen?
- Wie kann man diese Informationen extrahieren?
- Was kann man mit diesen Informationen in Folge machen?

Aus dieser Fragestellung heraus enstanden folgende Anforderungen:

1. Man benötigt eine sortierte Punktemenge der Kontur des Objektes, um weitere geometrische Zusammenhänge zu erhalten.
2. Man benötigt eine Information über die Eckpunkte des Objektes, um einerseits Begrenzungspunkte zu erhalten und andererseits zu wissen zwischen welchen Punkten man eine Linien-, Kreis- oder Kurvenanpassung durchführen kann.

3. Man benötigt eine Erkennung von Linien oder Kurven, um die Form des Objektes festzuhalten.

4. Man benötigt statistische Methoden zur Korrektur von Ausreißern.

5. Man benötigt Schnittverfahren von Linien, Kreisen oder Kurven um eindeutige Eckpunkte festzulegen.

Die Herausforderung bestand jetzt darin all diese Anforderungen so zu kombinieren, um wirklich zu einem brauchbaren Ergebnis zu kommen.

1.3.3 Informationsfluss

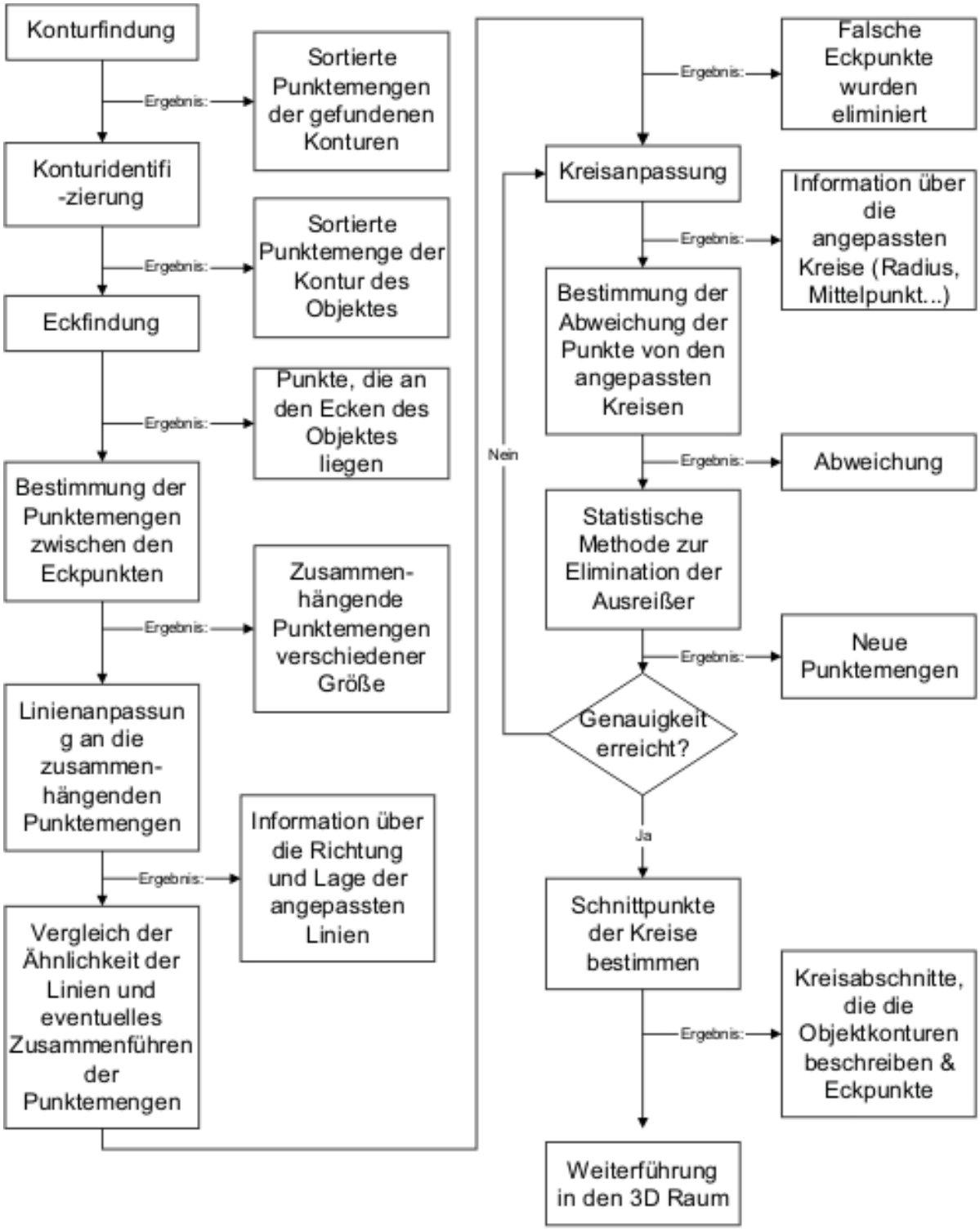


Abbildung 1.5: Informationsfluss

Kapitel 2

Das Rechenmodell

2.1 Allgemeines

Da es in der Geometrie für unsere Zwecke sinnvoll ist homogene Koordinaten zu verwenden, möchte ich noch kurz darauf eingehen. Zu diesem Zwecke ist es angebracht, hier einige Erörterungen über gewisse geometrische Vorstellungen darzustellen, die man mit den homogenen Koordinaten verbinden kann. Man spricht da zunächst nur von einer Ebene E. In diesem Falle setzt man für die beiden rechtwinkligen Koordinaten:

$$x = \frac{\xi}{\tau}, \qquad y = \frac{\eta}{\tau}.$$

Man will nun ξ, η und τ als rechtwinklige Koordinaten eines Raumes deuten und in diesem Raum die Parallelebene $\tau = 1$ zur $\xi - \eta$ Ebene als die Ebene E auffassen (siehe Abbildung 2.1), indem man in ihr $x = \xi$ und $y = \eta$ setzt. Verbindet man nun den Punkt x, y von E mit 0 durch einen geradlinigen Strahl, so ist bekanntlich auf ihm $\frac{\xi}{\tau}$ und $\frac{\eta}{\tau}$ konstant und zwar muss:

$$\frac{\xi}{\tau} = x, \qquad \frac{\eta}{\tau} = y$$

sein, da ja für $\tau = 1$ gerade $\xi = x$ und $\eta = y$ sein sollte.

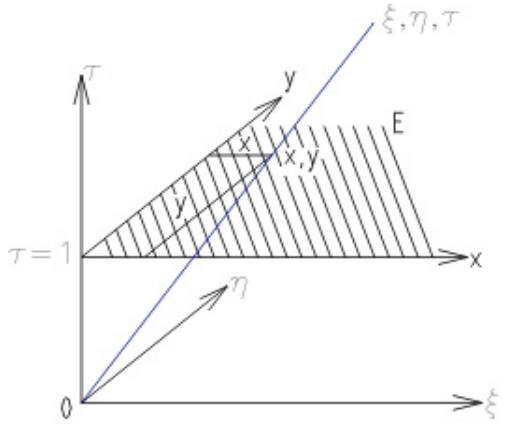


Abbildung 2.1: Homogene Koordinaten

Die Einführung homogener Koordinaten bedeutet hiernach einfach die Abbildung der Ebene E auf das sie aus dem Nullpunkt 0 des dreidimensionalen Hilfsraumes projizierende Strahlenbündel: Die homogenen Koordinaten eines Punktes sind die Raumkoordinaten der Punkte des ihn projizierenden Strahles dieses Bündels. [3]

Da jedem Punkt von E die unendlich vielen Punkte des Strahles zugehören, ist die Bedeutung der Unbestimmtheit der homogenen Koordinaten völlig klargestellt. Dass das Wertesystem $\xi = \eta = \tau = 0$ ausgeschlossen ist, hat seinen geometrischen Grund darin, dass durch den Punkt 0 allein noch kein Strahl und daher kein Punkt in E bestimmt ist. Auch dass man keine unendlichen Werte ξ, η und τ braucht, ist selbstverständlich, da man alle Strahlen bereits durch Verbindung von 0 mit im Endlichen gelegenen Punkten erhält. Und es wird anschaulich wie man unendlich große Werte der Koordinaten vermeidet, indem man das unendlich Ferne der Ebene durch die in $\tau = 0$ gelegenen Parallelstrahlen durch 0 ersetzt.

Auch die bekannte Sprechweise von der unendlich fernen Geraden erhält hier einen anschaulichen geometrischen Inhalt.

Analytisch ist sie nur der Ausdruck der abstrakten Analogie, dass alle ,,unendlich fernen Punkte“ der linearen Gleichung $\tau = 0$ genügen, genau wie jede endliche Gerade eine lineare Gleichung besitzt. Jetzt kann man aber ganz geometrisch sagen:

Jeder Geraden von E gehört im Bündel 0 ein ebenes Strahlenbüschel zu, und umgekehrt bestimmt jedes ebene Strahlenbüschel im Bündel 0 eine Gerade in E (abgesehen von dem ebenen Büschel $\tau = 0$).

Darum wird hier auch in der Schreibweise unterschieden:
Gegeben ist ein Punkt P_1 mit den Koordinaten x_1 und $y_1 \Rightarrow$

1. Affine Koordinaten dieses Punktes: $P_1 = [x_1, y_1]$
2. Homogene Koordinaten dieses Punktes: $P_1 = [x_1 : y_1 : 1]$

2.2 Konturfindung

2.2.1 Ziel

Das Ziel einer Konturfindung liegt darin, bei einem gegebenen Bild abhängig von einem gewissen Schwellwert Konturen zu finden. Die Konturfindung soll weiters so erfolgen, sodass die gefundene Punktemenge der Kontur schon sortiert ist.

2.2.2 Theorie, Lösungsweg, verwendete Algorithmen

Das Bild wird nun als Matrize eingelesen und folgendermaßen bearbeitet:
Grundsätzlich ist jeder Pixel von vielen anderen Punkten umgeben, jedoch beschränke ich meine Sicht nur auf die vier naheliegendsten Punkte. Diese stellen die Umgebung dar. Es wird bei jedem einzelnen Pixel, dessen Graustufenwert über dem Schwellwert liegt, eine Überprüfung der Umgebung durchgeführt. Es wird der Wert der umgebenden Pixel überprüft, ob dieser nun unter oder gleich dem Schwellwert ist. Ist das der Fall wird zwischen der aktuellen Pixelposition und dem Nachbarpixel (er wird ab hier als Treffer bezeichnet) die Position eines Konturpunktes interpoliert (ist der Wert des Treffers gleich dem Schwellwert fällt die Interpolation natürlich weg).

Es treten genau 16 verschiedene Treffermöglichkeiten auf, von denen 2 als Sonderfälle zu betrachten sind.

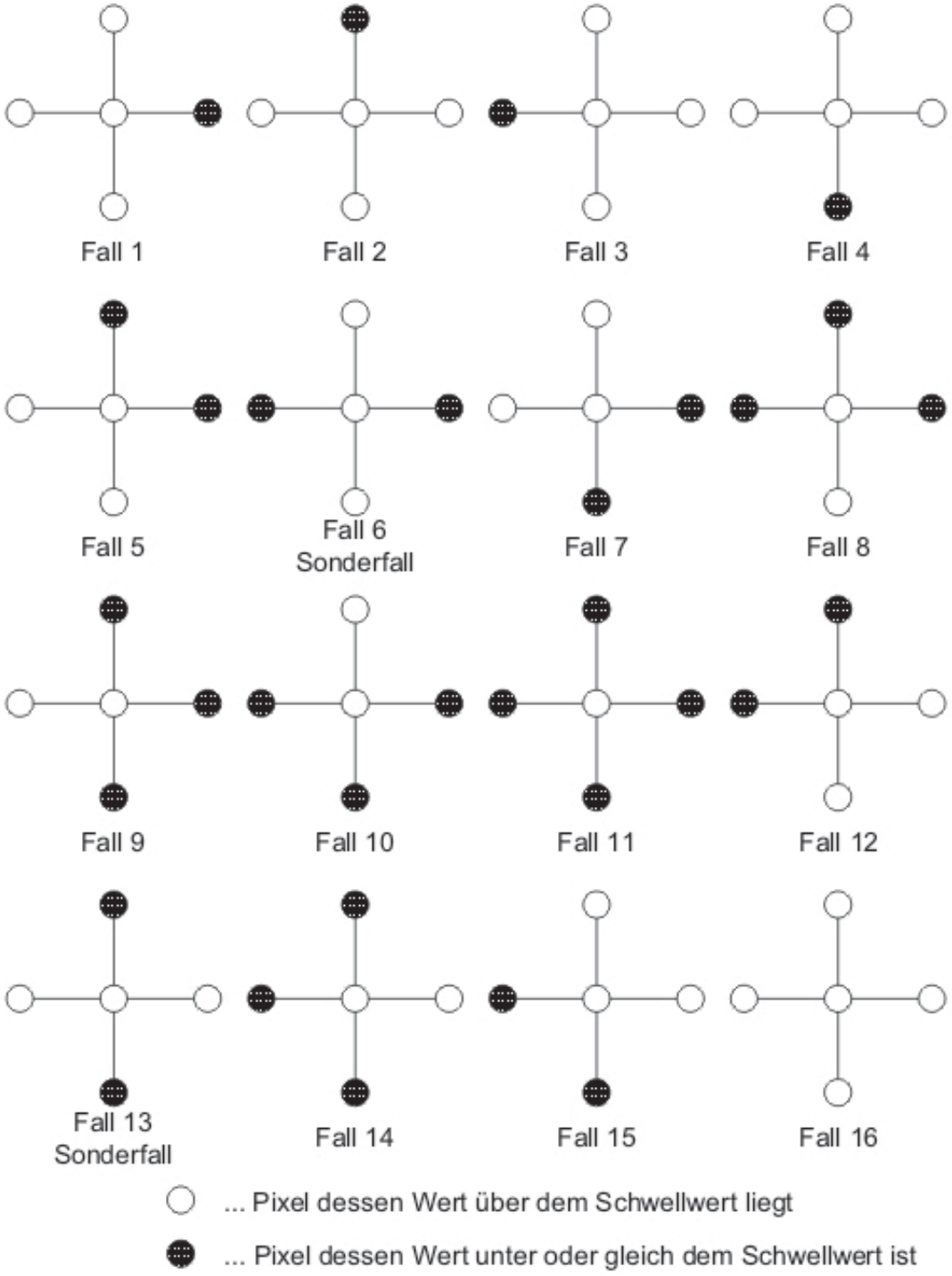


Abbildung 2.2: Treffermöglichkeiten

Der Fall 16 ist weiter auch nicht zu berücksichtigen, da hier in der Umgebung des Pixels keine Treffer gefunden wurden. Das besondere an den Sonderfällen liegt darin, dass es sich hier um die Konturpunkte zweier verschiedener Konturen handeln kann oder nicht. Jeder Sonderfall kann nun in 6 weitere Möglichkeiten gegliedert werden:
Die Position des Sonderfalles wird das erste Mal überprüft

1. Der Sonderfall wird zum ersten Mal durchlaufen und ist der erste Treffer in der Kontur und somit auch der letzte Treffer und diese Position wird zwischendurch nicht mehr durchlaufen, wobei dann nur ein Treffer berücksichtigt wurde und die Position nochmals überprüft werden muss. ⇒ Position wurde zweimal durchlaufen, beinhaltet aber noch einen Konturpunkt einer anderen Kontur. ⇒ Übergang zu 5 oder 6.

2. Der Sonderfall wird zum ersten Mal durchlaufen und ist der erste Treffer in der Kontur und somit auch der letzte Treffer und diese Position wird zwischendurch nochmals durchlaufen, wobei dann alle zwei Treffer berücksichtigt wurden und die Position nicht nochmals überprüft werden muss. ⇒ Position wurde dreimal durchlaufen und braucht nicht mehr überprüft werden.

3. Der Sonderfall wird zum ersten Mal durchlaufen, ist aber nicht der erste Punkt der aktuellen Kontur und wird in der aktuellen Kontur nicht mehr durchlaufen, wobei dann nur ein Treffer berücksichtigt wurde und die Position nochmals überprüft werden muss. ⇒ Position wurde einmal durchlaufen, beinhaltet aber noch einen Konturpunkt einer anderen Kontur. ⇒ Übergang zu 5 oder 6.

4. Der Sonderfall wird zum ersten Mal durchlaufen, ist aber nicht der erste Punkt der aktuellen Kontur und wird in der aktuellen Kontur nochmals durchlaufen, wobei dann zwei Treffer berücksichtigt wurden und die Position nicht nochmals überprüft werden muss. ⇒ Position wurde zweimal durchlaufen und braucht nicht mehr überprüft werden.

   Die Position des Sonderfalles wird das zweite Mal überprüft

5. Beim neuerlichen Überprüfen der Position ist der Treffer der Startpunkt einer Kontur und somit auch noch der Endpunkt. ⇒ Position wurde jetzt bei Fall 1 und 5 insgesamt viermal und bei Fall 3 und 5 insgesamt dreimal überprüft und ist somit abgeschlossen.

6. Beim neuerlichen Überprüfen der Position ist der Treffer nicht der Startpunkt einer Kontur und somit wird er nur einmal in die neue Kontur eingehen. ⇒ Position wurde jetzt bei Fall 1 und 6 insgesamt dreimal und bei Fall 3 und 5 insgesamt zweimal überprüft und ist abgeschlossen.

Um das ganze jetzt noch besser verstehen zu können kann man Abbildung 2.3 betrachten:

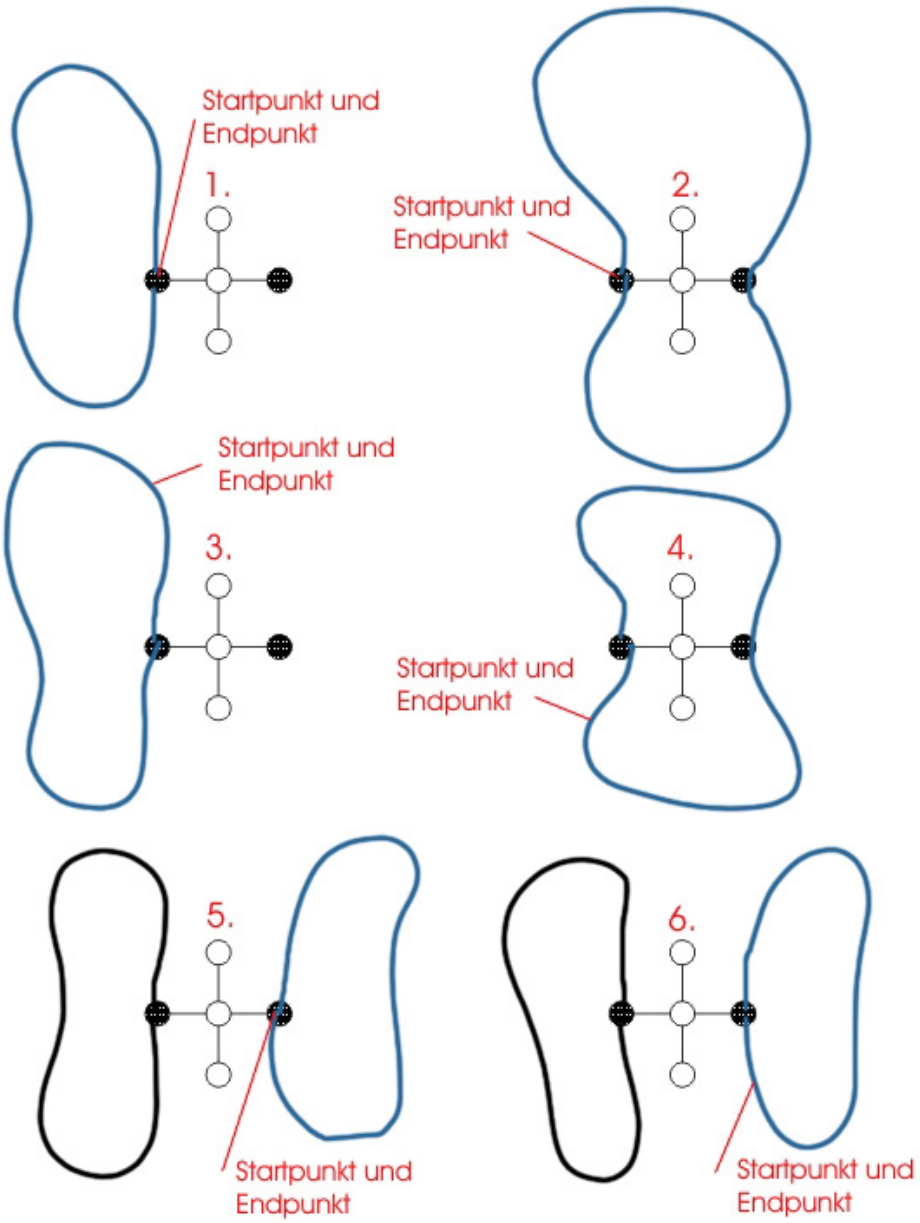


Abbildung 2.3: Sonderfälle

Bei all diesen 16 Möglichkeiten werden nun immer die Konturpunkte interpoliert und eine von vier möglichen logischen Suchrichtungen angegeben, die der Kontur folgen. Somit werden alle Konturpunkte nacheinander gefunden und sind auch schon sortiert. Es kann aber passieren, dass eine oder mehrere Konturen nicht geschlossen sind und die Suche nach der Kontur bis an den Rand des Bildes erfolgt. Wenn das passiert wird einfach an den ersten Punkt der Kontur zurückgegangen und die Suchrichtung invertiert und weitergesucht.

An dieser Stelle soll noch kurz auf die Interpolation zwischen den Pixeln eingegangen werden: Dadurch, dass man sich auf die Umgebung von 4 Pixel beschränkt, erhält man auch für 4 verschiedene Richtungen (Abbildung 2.4)

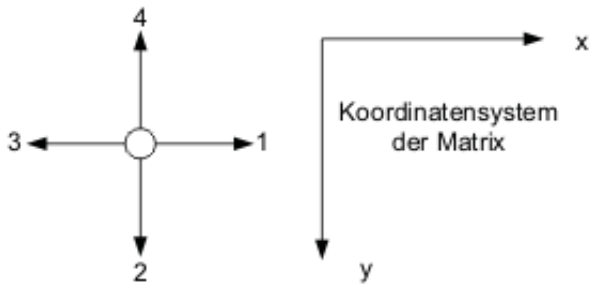


Abbildung 2.4: Richtungsmöglichkeiten und Koordinatensystem

die Gleichungen für die Interpolation:

- Richtung 1

$$PKX = APX + \frac{AW - SW}{AW - RW} \tag{2.1}$$

$$PKY = APY \tag{2.2}$$

- Richtung 2

$$PKX = APX \tag{2.3}$$

$$PKY = APY + \frac{SW - RW}{AW - RW} - 1 \tag{2.4}$$

- Richtung 3

$$PKX = APX + \frac{SW - AW}{AW - RW} \tag{2.5}$$

$$PKY = APY \tag{2.6}$$

- Richtung 4

$$PKX = APX \tag{2.7}$$

$$PKY = APY + \frac{RW - SW}{AW - RW} + 1 \tag{2.8}$$

PKX ... Position der Kontur in x-Richtung
APX ... Aktuelle Position in x-Richtung
PKY ... Position der Kontur in y-Richtung
APY ... Aktuelle Position in y-Richtung
AW ... Wert an der aktuellen xy-Position
SW ... Schwellwert
RW ... Wert an der xy-Position des Treffers

2.2.3 Beispiel

Als Beispiel wird hier ein 3D-Block dargestellt, bei dem gewisse Teile abgebrochen sind, die man links im Bild erkennen kann. Aufgrund derselben Objekteigenschaften (Temperatur) werden natürlich auch Konturen gefunden (siehe Abbildung 2.5).

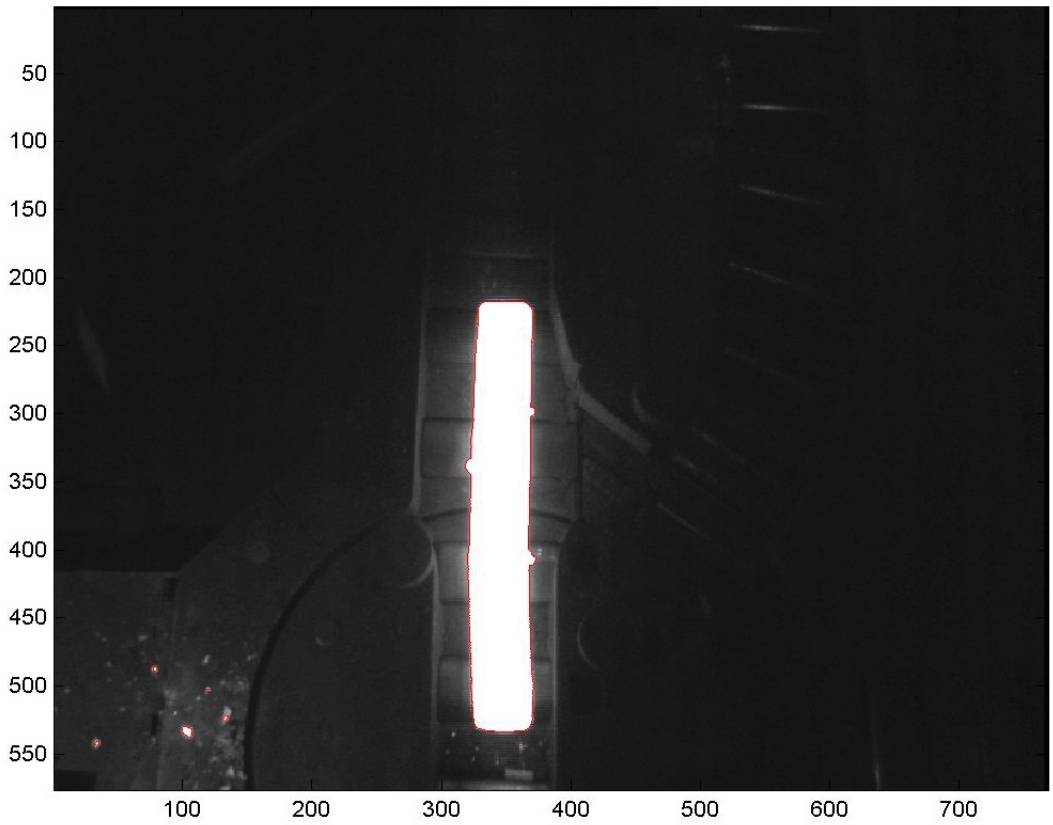


Abbildung 2.5: Konturbeispiel

Man kann jetzt gut erkennen, dass die Wahl des Schwellwertes wichtig ist um die äusserste Kontur des Objektes zu erfassen.

2.3 Konturidentifizierung

2.3.1 Ziel

Das Ziel liegt jetzt darin die richtige Kontur des Blockes zu identifizieren.

2.3.2 Theorie, Lösungsweg, verwendete Algorithmen

Es werden bei der Konturfindung viele kleine Konturen gefunden, die einerseits durch Reflexionen und andererseits durch abgebrochene Objektsplitter erzeugt werden, da die Objektsplitter natürlich die gleiche Temperatur wie das Objekt selber und somit auch die gleichen Helligkeitswerte haben. Es werden also außer der Kontur des Objektes noch weitere Konturen außerhalb (Splitter und Reflexionen) und innerhalb (Zunder) gefunden. Man geht jetzt von der Annahme aus, dass unser Objekt die längste Kontur besitzen muss. Wie kann man jetzt diese Kontur identifizieren? Es bietet sich eine Möglichkeit an, die den Flächeninhalt einer geschlossenen Kurve berechnet. Man betrachtet also eine geschlossene Kurve (siehe Abbildung 2.6) [3].

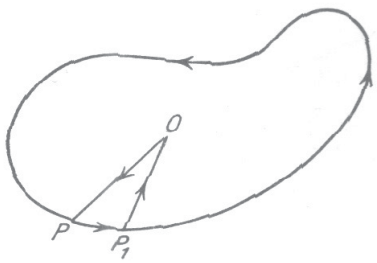


Abbildung 2.6: Konturidentifizierung Theorie 1

Man gibt auf ihr einen bestimmten Umlaufsinn an und fragt, welchen Inhalt sie begrenzt. Naturgemäß findet man ihn, wenn man die Kurve durch Polygone mit sehr vielen, sehr kleinen Seiten annähert und den Grenzwert bildet. Sind P(x,y) und $P_1(x + dx, y + dy)$ zwei benachbarte Eckpunkte eines solchen Näherungspolygons auf der Kurve, so setzt sich sein Inhalt aus einer Summe von Elemantardreiecken (O,P,P_1) zusammen, also aus lauter Summanden:

$$\frac{1}{2} \cdot \begin{vmatrix} 0 & 0 & 1 \\ \text{x} & \text{y} & 1 \\ \text{x+dx} & \text{y+dy} & 1 \end{vmatrix} = \frac{1}{2} \cdot (xdy - ydx) \tag{2.9}$$

In der Grenze geht diese Summe über in das längs der Kurve im gegebenen Sinn erstreckte Linienintegral:

$$\frac{1}{2} \cdot \oint (xdy - ydx) \tag{2.10}$$

und dadurch ist der von der Kurve begrenzte Flächeninhalt definiert. Will man das geometrisch interpretieren, so kann man das für Polygone ausgesprochene Resultat auf den neuen Fall sofort übertragen:
Jedes von der Kurve umschlossene Flächenstück kommt in dem Integral so oft positiv zur Geltung, als es entgegen dem Uhrzeigersinn, und so oft negativ, als es im Uhrzeigersinn umlaufen wird, während man die Kurve einmal im vorgeschriebenen Sinn durchläuft. [3]
Bei einer einfachen Kurve so wie in Abbildung 2.6 erhält man demnach durch das Integral genau den von ihr umschlossenen Flächeninhalt mit positivem Vorzeichen. In Abbildung 2.7

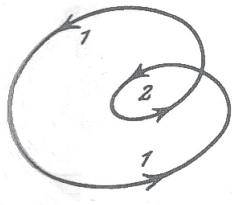


Abbildung 2.7: Konturidentifizierung Theorie 2

ist der äußere Teil einmal, der innere zweimal positiv gerechnet, in Abbildung 2.8

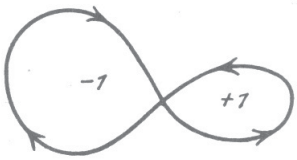


Abbildung 2.8: Konturidentifizierung Theorie 3

der linke negativ, der rechte positiv, so dass im ganzen ein negativer Inhalt herauskommt. In Abbildung 2.9

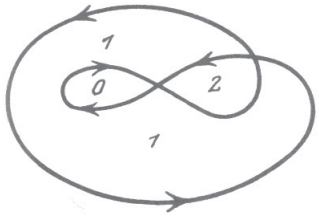


Abbildung 2.9: Konturidentifizierung Theorie 4

ist sogar ein Teil gar nicht zu rechnen, da er einmal positiv, einmal negativ umlaufen ist. Natürlich können so auch Kurven vom Inhalt Null entstehen. In unserem Fall jedoch wird niemals eine Kurve entstehen, die in sich Schnittpunkte aufweist, da das unserer Konturfindung widerspricht. Diese würde in diesem Fall nicht eine sondern mehrere Konturen finden.

2.3.3 Beispiel

Als Beispiel wird hier ein 3D-Block dargestellt, bei dem gewisse Teile abgebrochen sind, die man links im Bild erkennen kann. Aufgrund derselben Objekteigenschaften (Temperatur) werden natürlich auch dort Konturen gefunden (siehe Abbildung 2.5). Die Konturidentifizierung liefert nun folgendes Ergebnis (Abbildung 2.10).

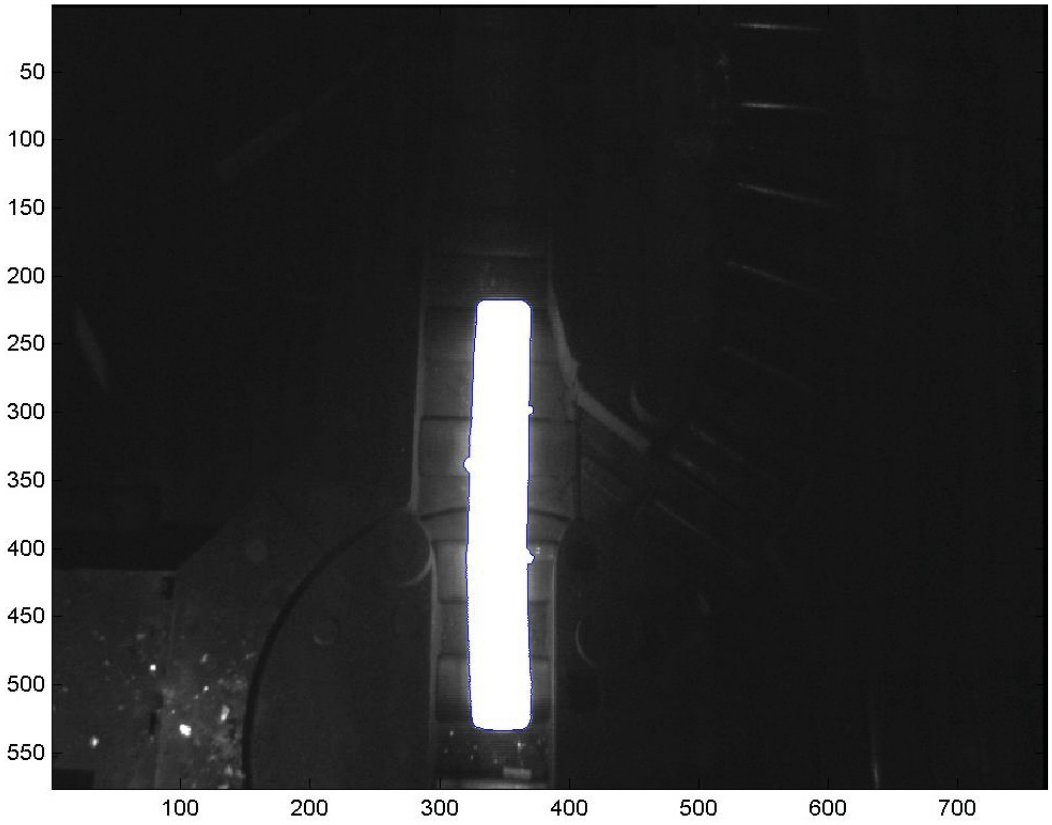


Abbildung 2.10: Konturidentifizierungsbeispiel

Das Ergebnis ist in diesem Bild mit einer blauen Linie gekennzeichnet.

2.4 Eckfindung

2.4.1 Ziel

Das Ziel liegt jetzt darin Punkte zu finden, die mit sehr großer Wahrscheinlichkeit an den Ecken des Objektes liegen.

2.4.2 Theorie, Lösungsweg, verwendete Algorithmen

Es gibt jetzt mehrere Möglichkeiten, diese Punkte zu finden, ich möchte hier aber 2 davon vorstellen.

- Die Anwendung der Grassmann Determinanten

  Wir betrachten zunächst drei aufeinanderfolgende Punkte in der Kontur:

  $$\begin{aligned} P_{i-1} &= [x_{i-1} : y_{i-1} : 1] \\ P_i &= [x_i : y_i : 1] \\ P_{i+1} &= [x_{i+1} : y_{i+1} : 1] \end{aligned}$$

  Das von diesen drei Punkten eingeschlossene Dreieck hat einen Flächeninhalt von

  $$A_i = \frac{1}{2} \cdot \begin{vmatrix} x_{i-1} & y_{i-1} & 1 \\ x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \end{vmatrix} \tag{2.11}$$

  Die Länge der Basislinie dieses Dreiecks lässt sich wie folgt berechnen:

  $$l_i = \sqrt{(x_{i+1} - x_{i-1})^2 + (y_{i+1} - y_{i-1})^2} \tag{2.12}$$

  Daraus folgt, dass der Abstand d_i des Mittelpunktes des Dreieckes zur Basislinie folgenden Wert hat:

  $$d_i = \frac{A_i}{l_i} \tag{2.13}$$

  Wobei darauf geachtet werden muss, dass die Länge einen minimalen Wert nicht unterschreitet. Nun kann dieser gefundene Wert d_i zur Abschätzung ob es sich nun um einen Konturpunkt handelt, herangezogen werden. Um das ersichtlich zu machen, warum man den Wert d_i als Schlüsselwert verwenden kann folgende 2 Beispiele (Abbildung 2.11 und 2.12):

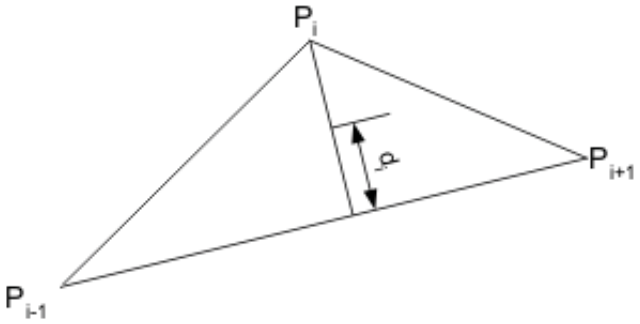


Abbildung 2.11: Entscheidungskriterium d_i bei der Eckfindung

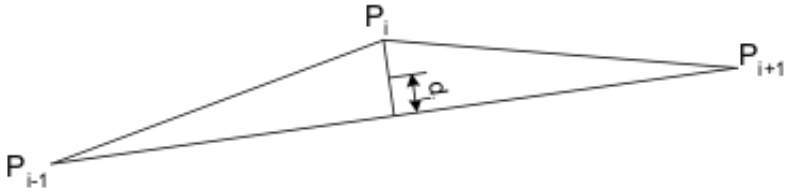


Abbildung 2.12: Entscheidungskriterium d_i bei der Eckfindung

Man kann jetzt sehr gut erkennen, dass sich d_i als Entscheidungsmerkmal eignet, da es bei flacherem Kurvenverlauf auch kleiner ist und es somit unwahrscheinlich ist, dass es sich um einen Eckpunkt handelt. Diese Methode eignet sich aber für unsere Anwendung nicht, da der Abstand der Konturpunkte ziemlich gering ist und sich aufgrund der einigermaßen großen Helligkeitsschwankungen im Pixelbereich ein Zickzackkurs der Kontur ergibt.

- Grassmannian Reduktion [2]

  Man geht hier von der Idee aus, dass ein Punkt ein Eckpunkt ist, wenn man einen Kreis einpassen kann, bzw wenn das Einpassen einer Linie zu einem großen Fehler führt. Man zieht einen Kreis heran, da sich Ecken bei Vergrößern der einzelnen Bilder als abgerundete Ecken bzw. als Kreissegmente darstellen.

  Die allgemeine Kreisgleichung lautet:

$$(x - x_0)^2 + (y - y_0)^2 - r^2 = 0 \tag{2.14}$$

  Man multipliziert die Gleichung nun aus und und fasst sie in einzelne Terme zusammen:

$$x^2 + y^2 - 2x_0x - 2y_0y + x_0^2 + y_0^2 - r^2 = 0 \tag{2.15}$$

Grassman schlägt in der Ausdehnungslehre vor, anstatt der Variablen selbst, Parameter zur Beschreibung des Bereiches zu nutzen der analysiert werden soll. Der Parameterraum hat normalerweise eine höhere Ordnung als der Variablenraum. Nun werden die Koeffizienten der obigen Gleichung als Grassmann Koordinaten verwendet:

$$C_1(x_i^2 + y_i^2) - C_2 x_i + C_3 y_i - C_4 = 0 \tag{2.16}$$

Sehr wichtig ist es nun, dass diese Gleichung linear ist und mit linearer Algebra gelöst werden kann. Wie man weiß genügen 3 Punkte ($P_1 = [x_1 : y_1 : 1]$ $P_2 = [x_2 : y_2 : 1]$ $P_3 = [x_3 : y_3 : 1]$), um einen Kreis zu beschreiben. Ein beliebiger Punkt P=[x:y:1] liegt nun auf den von den ersten 3 Punkten definierten Kreis, wenn folgende Bedingung erfüllt ist:

$$\begin{vmatrix} x^2 + y^2 & x & y & 1 \\ x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ x_3^2 + y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = 0 \tag{2.17}$$

Man betrachtet jetzt aber nicht nur einen Bereich von 3 Punkten, sondern einen Bereich von 15 Punkten und berechnet die Koeffizienten mit der **Singular Value Decomposition**. Die Gleichung lautet nun folgendermaßen, da die Punkte eben nicht genau auf einer Kreisgleichung liegen:

$$C_1(x_i^2 + y_i^2) - C_2 x_i + C_3 y_i - C_4 = e_i \neq 0 \tag{2.18}$$

Matrizenform:

$$\begin{bmatrix} x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ x_3^2 + y_3^2 & x_3 & y_3 & 1 \\ x_4^2 + y_4^2 & x_4 & y_4 & 1 \\ x_5^2 + y_5^2 & x_5 & y_5 & 1 \\ x_6^2 + y_6^2 & x_6 & y_6 & 1 \\ x_7^2 + y_7^2 & x_7 & y_7 & 1 \\ x_8^2 + y_8^2 & x_8 & y_8 & 1 \\ x_9^2 + y_9^2 & x_9 & y_9 & 1 \\ x_{10}^2 + y_{10}^2 & x_{10} & y_{10} & 1 \\ x_{11}^2 + y_{11}^2 & x_{11} & y_{11} & 1 \\ x_{12}^2 + y_{12}^2 & x_{12} & y_{12} & 1 \\ x_{13}^2 + y_{13}^2 & x_{13} & y_{13} & 1 \\ x_{14}^2 + y_{14}^2 & x_{14} & y_{14} & 1 \\ x_{15}^2 + y_{15}^2 & x_{15} & y_{15} & 1 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \\ e_8 \\ e_9 \\ e_{10} \\ e_{11} \\ e_{12} \\ e_{13} \\ e_{14} \\ e_{15} \end{bmatrix} \tag{2.19}$$

Vereinfachte Schreibweise:

$$\mathbf{G} \cdot \mathbf{c} = \mathbf{e} \tag{2.20}$$

Das Ziel liegt nun darin die Parameter $C_1 : C_2 : C_3 : C_4$ zu bestimmen. Hierzu müsste man die Matrix G invertieren, diese ist aber nicht quadratisch, deshalb greift man zur schon bereits erwähnten **Singular Value Decomposition** [5] [1]. Dabei bedeutet die Gleichung 2.20, dass der Nullraum der Matrize G gesucht wird. Die „Singular Values“, die im Deutschen als „singuläre Werte“bezeichnet werden, sind eine Norm für die Distanz des jeweiligen Vektors a vom Nullraum. Die Säulenvektoren der in 2.21 berechneten Matrix v entsprechen den kleinsten „Singular Values“und sind die gesuchte Lösung des Gleichungssystems nach 2.19 und 2.20.

In **MATLAB** kann diese Gleichung folgend gelöst werden:

$$\mathbf{[u,s,v] = SVD(G);} \tag{2.21}$$

wobei mit

$$\mathbf{c = v(:,end)}; \tag{2.22}$$

die Parameter $C_1 : C_2 : C_3 : C_4$ gefunden werden

Überschreitet der Koeffizient C_1 nun einen gewissen Schwellwert wissen wir, dass das Kurvenstück einen gewissen Kreisanteil hat und nicht nur aus einer Linie besteht. Trifft das nun zu, wird der mittlere Punkt des Kurvenstücks als Eckpunkt markiert.

Man kann nun im Falle einer Linie die ,,planner line“ Koordinaten bzw. die Determinanten X, Y, N wie folgt ermitteln:

$$X = -C_3 \tag{2.23}$$
$$Y = C_2 \tag{2.24}$$
$$N = C_4 \tag{2.25}$$
$$\tag{2.26}$$

Im Falle eines Kreises lassen sich die Mittelpunktskoordinaten und die Radien auch aus den Koeffizienten ermitteln:

$$x_0 = -\frac{C_2}{2C_1} \tag{2.27}$$
$$y_0 = -\frac{C_3}{2C_1} \tag{2.28}$$
$$r = \sqrt{x_0^2 + y_0^2 - \frac{C_4}{C_1}} \tag{2.29}$$

2.4.3 Beispiel

Als Beispiel wird hier ein 3D-Block dargestellt, bei dem gewisse Teile abgebrochen sind, die man links im Bild erkennen kann. Die Eckidentifizierung liefert nun folgendes Ergebnis (Abbildung 2.13).

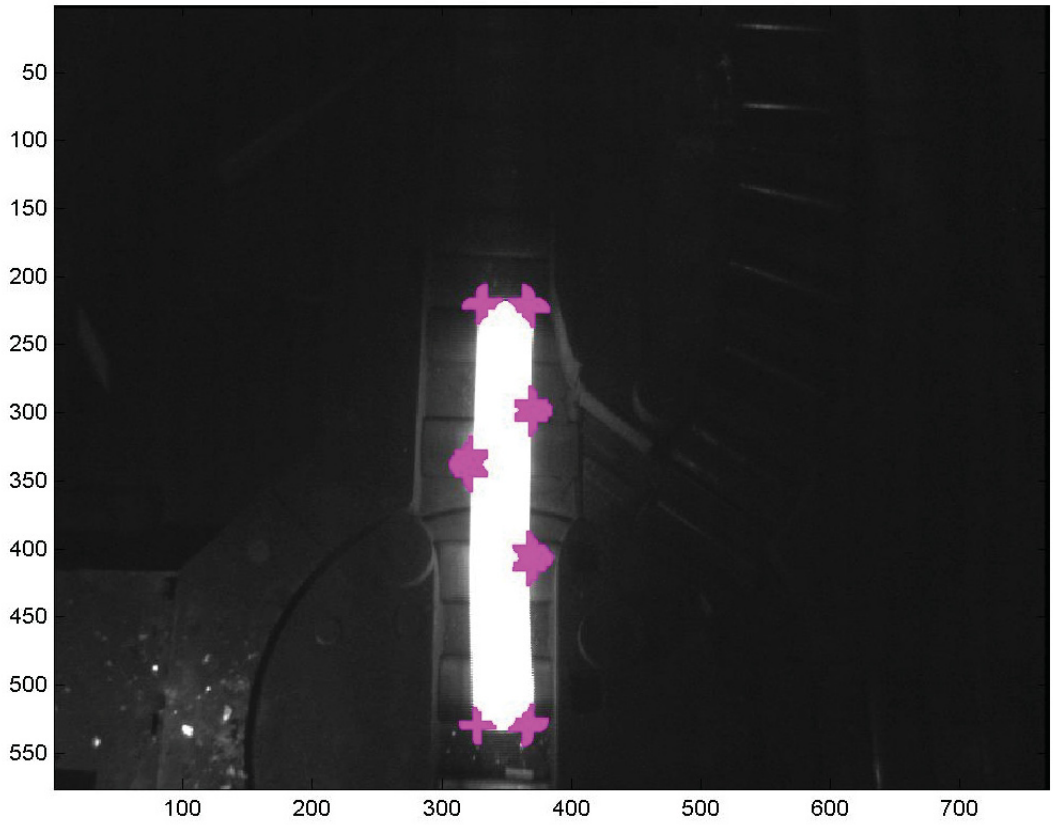


Abbildung 2.13: Eckidentifizierungsbeispiel

Das Ergebnis ist in diesem Bild mit lila Markern gekennzeichnet. Man kann sehr gut erkennen, dass das Programm viele Eckpunkte findet, die eigentlich keine Eckpunkte sind. Aus diesem Grund soll nun mit einer Linienanpassung zwischen den Ecken und dem Vergleich dieser Linien eine Methode gefunden werden, mit der man falsche Eckpunkte eliminiert.

2.5 Linienanpassung

2.5.1 Ziel

Es geht nun darum, zwischen eine Menge von Punkten eine Linie einzupassen, sodass die Punkte möglichst nahe an der Linie liegen.

2.5.2 Theorie, Lösungsweg, verwendete Algorithmen

Das Grassmannsche Determinantenprinzip für Linien in einer Ebene [3]
Die Determinante aus den Koordinaten dreier Punkte ergibt, wie schon erwähnt den doppelten Flächeninhalt des von den drei Punkten eingeschlossenen Dreiecks:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \tag{2.30}$$

Nunmehr will man außerdem auch die aus den Koordinaten zweier bzw. eines Punktes gebildeten Schemata:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} \tag{2.31}$$

bzw.

$$\begin{vmatrix} x_1 & y_1 & 1 \end{vmatrix} \tag{2.32}$$

betrachten, die als Matrizen bezeichnet werden. Jede solche Matrix soll die *Gesamtheit der Determinanten repräsentieren, die sich aus ihr durch Fortlassen einer bzw. zweier Kolonnen ergeben* . So erhält man aus der ersten Matrix durch Fortlassen der ersten bzw. zweiten Kolonne die zweireihigen Determinanten:

$$Y = y_1 - y_2 \tag{2.33}$$

$$X = x_1 - x_2 \tag{2.34}$$

und durch Fortlassen der dritten ebenso:

$$N = x_1 y_2 - x_2 y_1 \tag{2.35}$$

Die Bezeichnungen sind so gewählt, wie es sich für die Raumgeometrie als zweckmäßig erweisen wird. Es ist zu untersuchen, was für ein geometrisches Gebilde durch diese 3 Determinanten X, Y, N festgelegt wird. X, Y und N werden auch als die ,,planner line coordinates“ bezeichnet. Bei der zweiten einreihigen Matrix entstehen als einreihige Determinanten neben der Zahl 1 die Koordinaten x_1, y_1 selbst. Sie bestimmen den Punkt mit diesen Koordinaten als einfachste Elementargröße und verlangen keine weitere Untersuchung mehr. Danach ist das Grassmannsche Prinzip einfach auszusprechen: *Es sollen in der Ebene sowie im Raum alle Matrizen (mit weniger Zeilen als Kolonnen) betrachtet werden, deren Zeilen je aus den Koordinaten eines Punktes und einer 1 gebildet sind. Es soll sodann untersucht werden, welche geometrischen Gebilde durch die Determinanten festgelegt sind, die man aus ihnen durch Wegstreichen hinreichend vieler Kolonnen erhält.* Nun aber wieder zu einem konkreten, ebenen Problem.

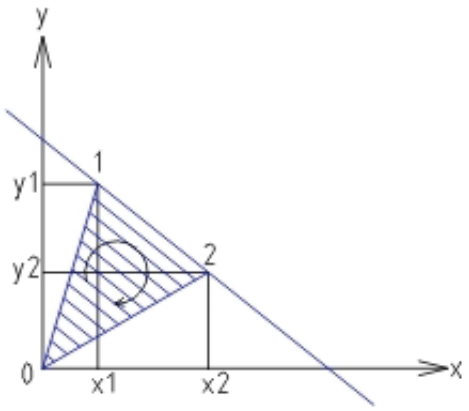


Abbildung 2.14: Das Grassmannsche Determinantenprinzip

Was ist an der Abbildung 2.14 gegeben, wenn man die Determinanten X, Y, N kennt? Offenbar bleibt dann in der Lage der Punkte noch eine Freiheit, da sie vollständig erst durch 4 Größen bestimmt ist. Dass durch X, Y, N zunächst die Verbindungsgerade 1 2 bestimmt ist, folgt unmittelbar aus der Tatsache, dass man ihre Gleichung:

$$\begin{vmatrix} \mathrm{x} & \mathrm{y} & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0 \tag{2.36}$$

auch in der Form schreiben kann:

$$Y \cdot x - X \cdot y + N = 0 \tag{2.37}$$

Hieraus erkennt man noch weiter, dass diese Gerade bereits durch die Verhältnisse X:Y:N bestimmt ist. Ferner kann man noch entnehmen, dass X und Y die Projektionen der Strecke(1,2) mit der Richtung von 2 nach 1 auf die x- und y-Achse sind, N aber den doppelten Inhalt

des Dreiecks(0,1,2) mit dem Umlaufsinn 0,1,2 darstellt. Offenbar ist daher die einzigen Lageänderungen der Punkte 1,2 bei denen diese drei Größen sämtlich unverändert bleiben, die Verschiebung der Strecke(1,2) unter Erhaltung ihrer Länge und ihres Sinnes längs ihrer Geraden. Man kann jetzt eine Gerade, die durch zwei Punkte definiert ist folgendermaßen anschreiben:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} Y \\ -X \\ N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.38}$$

Dabei liegen nun die Punkte genau auf der durch sie definierten Linie. Die Gleichung ist natürlich auch für eine beliebige Anzahl von Punkten erfüllt, die alle auf der Linie liegen:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} Y \\ -X \\ N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \tag{2.39}$$

Ist das nun aber nicht der Fall und die Punkte liegen so im Raum, dass die daraus gebildete Linie nur eine Annäherung ist, wird sich die obige Gleichung wie folgt ändern:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} Y \\ -X \\ N \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \\ \cdot \\ \cdot \\ e_n \end{bmatrix}, \qquad e_1 ... e_n \neq 0 \tag{2.40}$$

Man hat es nun vereinfacht wieder mit einem Gleichungssystem folgender Form zu tun,

$$\mathbf{A} \cdot \mathbf{c} = \mathbf{e} \tag{2.41}$$

in dem A nicht quadratisch ist und man zu den Determinanten X, Y, N wieder nur wie schon im vorigen Kapitel beschrieben durch die Singular Value Decomposition kommen kann. Das Ergebnis sind die drei Determinanten X, Y, N die eine Gerade vollständig beschreiben und somit ist das Ziel erreicht.

2.5.3 Beispiel

Als Beispiel wird hier ein 3D-Block dargestellt, bei dem gewisse Teile abgebrochen sind, die man links im Bild erkennen kann. Die Linienanpassung liefert nun folgendes Ergebnis (Abbildung 2.15).

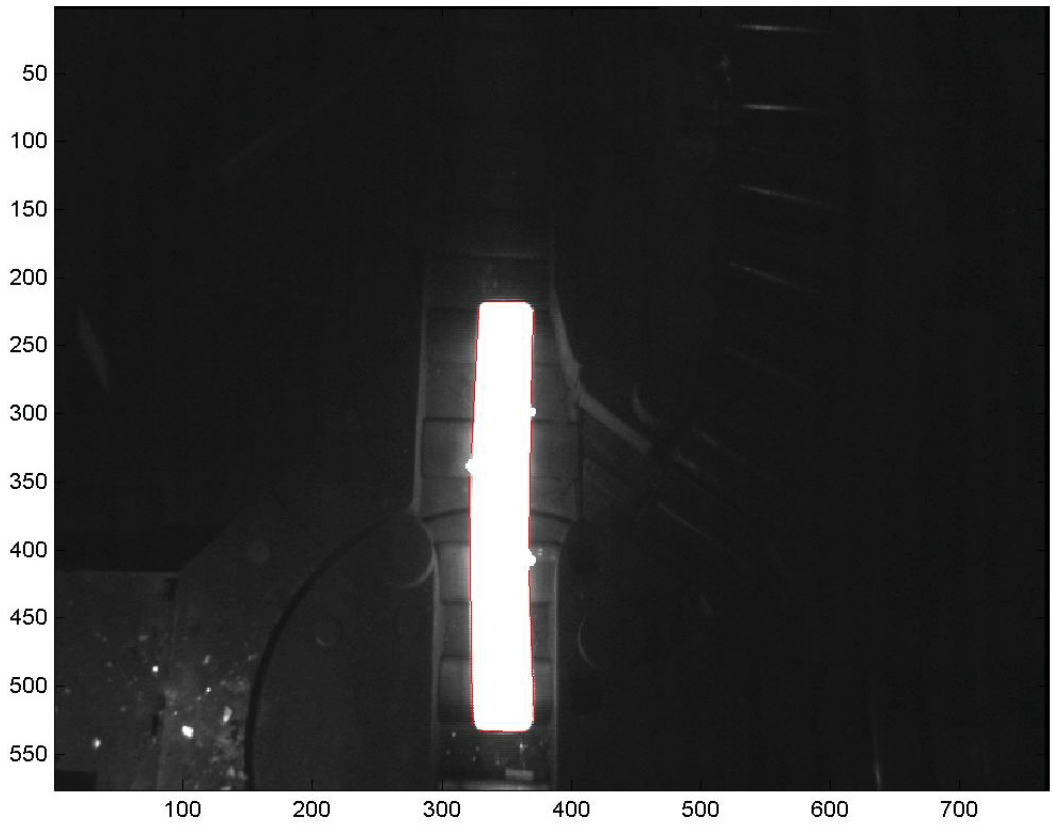


Abbildung 2.15: Linienanpassungsbeispiel

Die einzelnen Linien sind im Bild in rot eingezeichnet. Da die Linien genau zwischen den schon gefundenen Eckpunkten liegen, kann man diese nun dazu benutzen über Ähnlichkeiten falsche Eckpunkte zu elimineren und die einzelnen Datenmengen für die Linien- und Kreisanpassung extrahieren.

2.6 Ähnlichkeit von Geraden

2.6.1 Ziel

Das Ziel liegt darin, Ähnlichkeiten von Geraden zu bestimmen und die so gewonnenen Erkenntnisse zu nutzen.

2.6.2 Theorie, Lösungsweg, verwendete Algorithmen

Die Geradengleichung mit Plücker Koordinaten lautet, wie im vorigen Kapitel beschrieben:

$$Y \cdot x - X \cdot y + N = 0 \tag{2.42}$$

wobei die Determinanten X, Y und N schon bestimmt worden sind. Jetzt stellt sich uns die Frage, inwiefern man mit diesen Parametern eine Aussage über die Ähnlichkeit treffen kann und welche Ähnlichkeiten bestehen.

Hat man jetzt 2 Linien, die miteinander verglichen werden sollen,

$$l_1 \triangleq Y_1 x - X_1 y + N_1 = 0 \tag{2.43}$$
$$l_2 \triangleq Y_2 x - X_2 y + N_2 = 0 \tag{2.44}$$
$$\tag{2.45}$$

so ergeben sich die Schnittpunkte zu: siehe Kapitel 2.8.2 Gleichung 2.79 und 2.80.
Diese zwei Gleichungen kann man als Determinante anschreiben:

$$(X : Y : N) = \begin{vmatrix} X_1 & Y_1 & N_1 \\ X_2 & Y_2 & N_1 \end{vmatrix} \tag{2.46}$$

Daraus ergeben sich die homogenen Koordinaten für den Schnittpunkt als:

$$X = X_1 N_2 - X_2 N_1, \quad Y = Y_1 N_2 - Y_2 N_1, \quad N = X_1 Y_2 - X_2 Y_1, \tag{2.47}$$

und die affinen Koordinaten:

$$x = \frac{X}{N} \qquad y = \frac{Y}{N} \tag{2.48}$$

Betrachtet man jetzt die 2 Linien die mit den Plücker Koordinaten definiert sind als Punkte in einem homogenen 3D Raum, so definieren die Plücker Koordinaten der Linie, die durch diese zwei Punkte geht, die homogenen Koordinaten des Schnittpunktes. Es ergibt sich daraus der Vorteil, dass sofort erkannt werden kann wie die Linien zueinander liegen:

- Parallele Linien (siehe Abbildung 2.16) schneiden sich im Unendlichen, das in homogenen Koordinaten in N=0 resultiert.

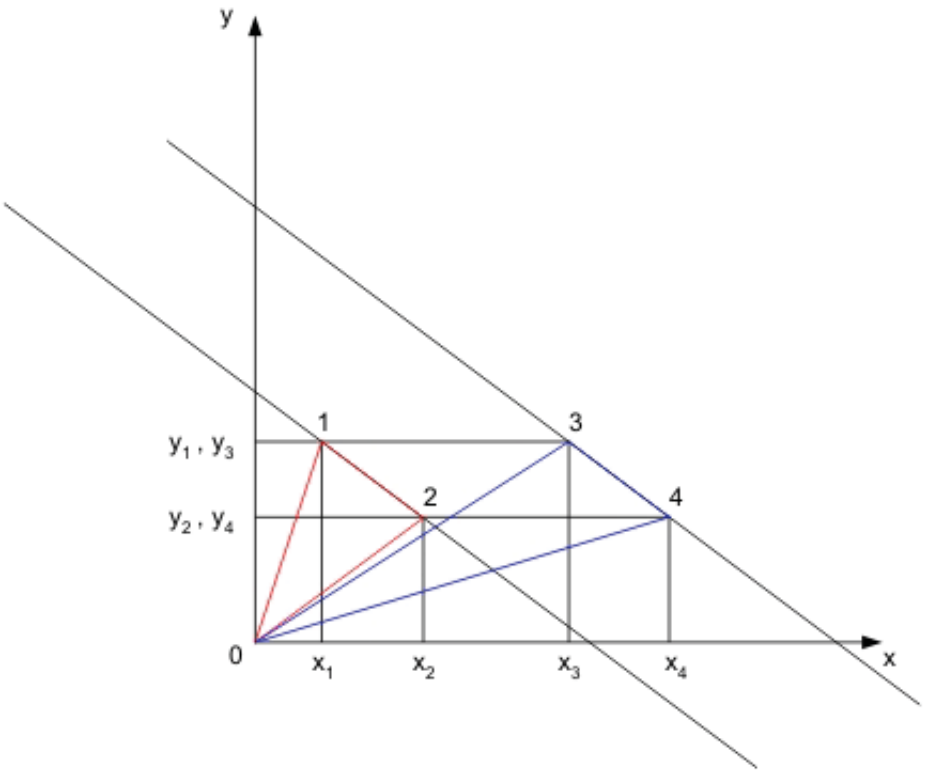


Abbildung 2.16: Parallele Linien

- Kolineare Linien liefern X=0, Y=0 und N=0.
- Für alle anderen Fälle können die affinen Koordinaten ermittelt werden.

Der Vergleich aufeinanderfolgender Linien wird in diesem Rechenmodell mit dem Vergleich von X_1 mit X_2, Y_1 mit Y_2 und N_1 mit N_2 durchgeführt.

Sind sich jetzt 2 Linien, zwischen denen Eckpunkte liegen, ähnlich, so kann man sagen, dass diese Eckpunkte eigentlich nur Ausreißer sind. Somit kann man falsche Eckpunkte eliminieren und es ergeben sich neue zusammengesetzte Datenmengen.

2.6.3 Beispiel

Als Beispiel wird hier ein 3D-Block dargestellt, bei dem gewisse Teile abgebrochen sind, die man links im Bild erkennen kann. Der Linienvergleich liefert nun folgendes Ergebnis (Abbildung 2.17).

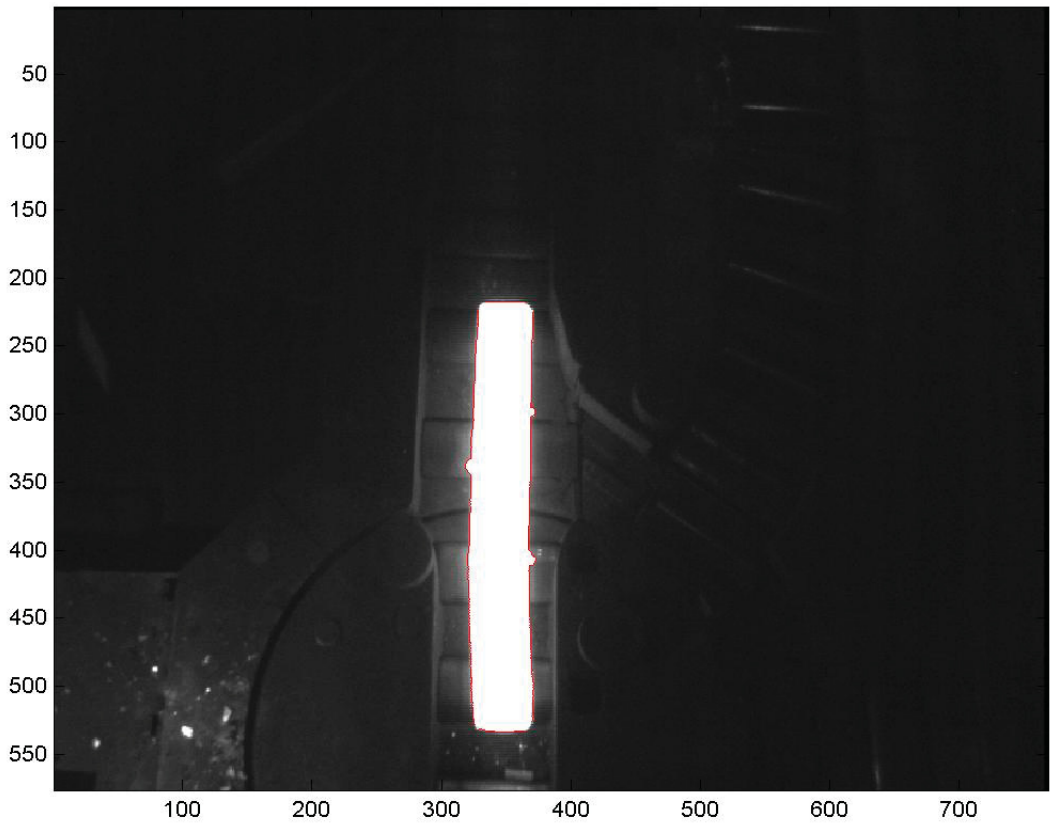


Abbildung 2.17: Linienvergleichsbeispiel

Die einzelnen neuen Datenmengen zwischen den Eckpunkten sind in Abbildung 2.17 in rot eingezeichnet.

2.7 Statistische Elimination der Ausreisser

2.7.1 Ziel

Das Ziel liegt darin Konturpunkte, die die Linien- bzw. Kreisanpassung in deren Genauigkeit negativ beeinflussen mit statistischen Methoden aus der Datenmenge herauszufiltern.

2.7.2 Theorie, Lösungsweg, verwendete Algorithmen

Grundsätzlich geht es jetzt darum aus einer Reihe von Werten, diejenigen zu finden, die von dem wahren Wert abweichen. Das führt uns zu den Begriffen von Wahrscheinlichkeitsdichte, Standard-Normalverteilung, Standardabweichung, Varianz, Mittelwert und wahren Wert. Die Statistik bezieht sich normalerweise auf eine Anzahl i von Werten x_i, die unter denselben immer wieder nachvollziehbaren Bedingungen bestimmt worden sind und einer gewissen natürlichen Streuung unterliegen. Von einer statistischen Analyse der Daten und der Fehlerquellen, die die Daten beeinflussen, kann man x' schätzen als

$$x' = \bar{x} \pm u_x(P\%) \tag{2.49}$$

wobei $\bar{x}$ die wahrscheinlichste Schätzung von x', basierend auf den vorhandenen Daten und u_x das Vertrauensintervall bei einer gewissen Wahrscheinlichkeit, repräsentiert.

Wahrscheinlichkeitsdichte [4]
Egal wie genau eine Messung durchgeführt wird um eine Anzahl von Daten zu erlangen, wird trotzdem eine gewisse zufällige Streuung auftreten. Diese Daten werden dann als Zufallsvariable bezeichnet. Bei einer wiederholten Messung von Daten unter gleichbleibenden Bedingungen wird jeder Datenwert dazu tendenzieren einen gewissen Wert zu haben oder innerhalb eines Intervalles zu diesem Wert zu liegen. Der zentrale Wert und die Werte, die darum verteilt sind können von der Wahrscheinlichkeitsdichte festgelegt werden. Das heisst das die Anzahl wie oft eine Zufallsvariable einen gewissen Wert, der rund um den zentralen Wert liegt hat, durch die Wahrscheinlichkeitsdichte beschrieben wird.

Betrachtet man nun ein Beispiel von diskreten Zufallsvariablen x, aufgelistet in der Tabelle 2.1, die aus N einzelnen Messungen x_i, i=1,2,..N besteht, wobei jede einzelne Messung während identischer Arbeitsbedingungen durchgeführt wurde.

i	x_i	**i**	x_i
1	0.98	11	1.02
2	1.07	12	1.26
3	0.86	13	1.08
4	1.16	14	1.02
5	0.96	15	0.94
6	0.68	16	1.11
7	1.34	17	0.99
8	1.04	18	0.78
9	1.21	19	1.06
10	0.86	20	0.96

Tabelle 2.1: Beispiel der Streuung einer Variable x

Die einzelnen Werte der Variablen kann man grafisch folgendermaßen auftragen (Abbildung 2.18):

Abbildung 2.18: Streuungsbeispiel

Man kann jetzt deutlich sehen, dass ein Gebiet entsteht, in dem sehr viele Werte dicht beieinanderliegen. Dieses Gebiet enthält nun den zentralen Wert. **So ein Verhalten ist typisch für die meisten gemessenen Werte in der Technik.** Außerdem können wir jetzt vermuten, dass der wahre Wert irgendwo innerhalb dieses Intervalles liegt. Man kann jetzt die Beschreibung der Variable x erweitern, indem man die Abszisse zwischen den maximalen und minimalen Wert von x in K Intervalle der Breite $\pm\delta x$ teilt. Die Anzahl der Werte, die jetzt in einem gewissen Intervall vorkommen wird auf der Ordinate aufgetragen.

$$K = 1.87(N-1)^{0.40} + 1 \tag{2.50}$$

Das resultierende Diagramm bezeichnet man nun als Histogramm der Variable x.

Das Histogramm (Abbildung 2.19) stellt eine Möglichkeit dar einerseits die Tendenz und andererseits die Dichte der Variable zu sehen. Wird nun die Ordinate so aufgetragen, dass die Anzahl der Variablen, die in den Intervallen liegen n_j durch die Gesamtanzahl der Variablen N dividiert wird, kommt man zur **Verteilungsfunktion**. Die Summe der Anzahl der

aufgetretenen Variablen,

$$\sum_{j=1}^{K} n_j = N$$

muss der Gesamtanzahl der Variablen N gleich sein.
Gleichermaßen muss gelten:

$$100 \times \sum_{j=1}^{K} f_j = 100\%$$

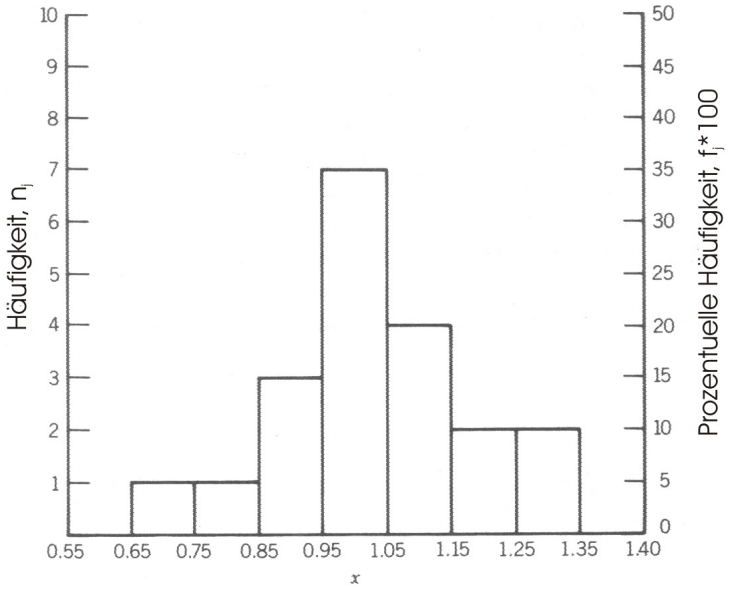


Abbildung 2.19: Histogramm

Die Wahrscheinlichkeitsdichtefunktion resultiert aus der Verteilungsfunktion bei $N \to \infty$ und $\delta x \to 0$.

$$p(x) = \lim_{N \to 0, \delta x \to 0} \sum_{j=1}^{K} \frac{n_j}{N \delta x} \tag{2.51}$$

Die Wahrscheinlichkeitsdichtefunktion bezieht den Messwert zu seiner Wahrscheinlichkeit des Auftretens. Die Wahrscheinlichkeitsdichtefunktion definiert die Wahrscheinlichkeit, dass eine gemessene Variable einen gewissen Wert annimmt und definiert auch noch den zentralen Wert. **Dieser zentrale Wert ist der gewünschte repräsentative Wert, der die beste Schätzung des wahren Wertes darstellt.** Ungeachtet der Art der Verteilung, kann eine Variable, die eine zentrale Tendenz hat, durch ihren **Mittelwert** und der Varianz beschrieben

werden. Der Mittelwert oder die zentrale Tendenz einer kontinuierlichen Zufallsvariablen x(t) mit der Wahrscheinlichkeitsdichtefunktion p(x) is definiert als

$$x' = \lim_{T\to\infty} \frac{1}{T} \int_0^T x(t)dt \tag{2.52}$$

Falls die Zufallsvariable von diskreten Werten gebildet wird gilt folgendes:

$$x' = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} x_i dt \tag{2.53}$$

Die Breite der Dichteverteilung spiegelt die Datenverteilung wieder. Für eine kontinuierliche Zufallsvariable wird die **Varianz** folgend definiert:

$$\sigma^2 = \lim_{T\to\infty} \frac{1}{T} \int_0^T [x(t) - x']^2 dt \tag{2.54}$$

oder für diskrete Werte

$$\sigma^2 = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} (x_i - x')^2 dt \tag{2.55}$$

Die **Standardabweichung** σ, ein weithin verbreiteter statistischer Parameter, wird definiert als die Wurzel der Varianz.

- Unendliche Statistik [4]

  Eine verbreitete Verteilung in technischen Bereichen ist die **Normal- oder Gaußverteilung**. Diese prognostiziert die Streuung symmetrisch rund um eine zentrale Tendenz. **Größen wie Länge, Temperatur, Druck und Geschwindigkeit werden meistens durch so eine Verteilung beschrieben, falls die Messungen unter den selben Arbeitsbedingungen durchgeführt wurden.** Streuung hervorgerufen durch Prozessunregelmäßigkeiten oder Genauigkeitsfehler sind üblicherweise normalverteilt. Die Form der Normalverteilung ist die weithin bekannte Glockenkurve (Abbildung 2.20).

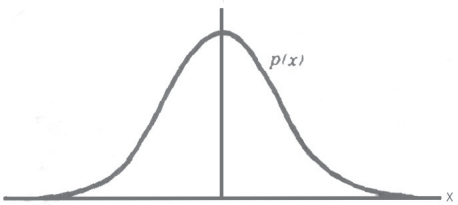


Abbildung 2.20: Normalverteilung

Und die dazugehörige Wahrscheinlichkeitsdichtefunktion errechnet sich aus

$$p(x) = \frac{1}{\sigma(2\pi)^{1/2}} exp[-\frac{1}{2}\frac{(x-x')^2}{\sigma^2}] \tag{2.56}$$

wobei x' als der wahre Mittelwert von x und σ^2 als die wahre Varianz von x definiert wird. Es gibt aber noch andere Wahrscheinlichkeitsdichtefunktionen, die ich hier kurz aufzählen möchte.

- Logarithmische Normalverteilung (Abbildung 2.21)
  Dient zur Ermittlung von Störungs- oder Halbarkeitsprognosen. Das sind Ereignisse, deren Ergebnisse dazu neigen, verzerrt zum äußeren Rand der Verteilung zu sein.

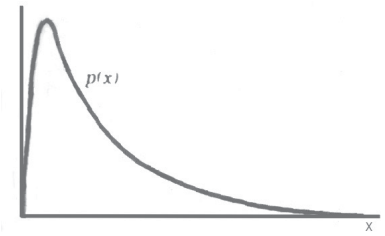


Abbildung 2.21: Logarithmische Verteilung

Und die dazugehörige Wahrscheinlichkeitsdichtefunktion errechnet sich aus

$$p(x) = \frac{1}{x\sigma(2\pi)^{1/2}} exp[-\frac{1}{2} ln \frac{(x-x')^2}{\sigma^2}] \tag{2.57}$$

- Poissonverteilung (Abbildung 2.22)
  Zufällig auftretende, zeitlich versetzte Ereignisse. P(x) bezieht sich auf die Wahrscheinlichkeit der Beobachtung von x Ereignissen in der Zeit t.

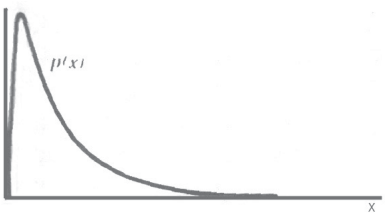


Abbildung 2.22: Poissonverteilung

Und die dazugehörige Wahrscheinlichkeitsdichtefunktion errechnet sich aus

$$p(x) = \frac{e^{-x}x^{xt}}{x!} \tag{2.58}$$

– Binomialverteilung (Abbildung 2.23)
Situationen, die die Häufigkeit von n eines bestimmten Ereignisses während D unabhängigen Tests beschreiben, wobei die Wahrscheinlichkeit P jedes Ereignisses die selbe ist.

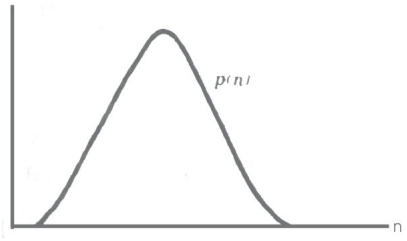


Abbildung 2.23: Binomialverteilung

Und die dazugehörige Wahrscheinlichkeitsdichtefunktion errechnet sich aus

$$p(n) = [\frac{N!}{(N-n)!n!}]P^n(1-P)^{N-n} \tag{2.59}$$

- Endliche Statistik [4]:

  Man hat jetzt eine Anzahl von Daten N. Ist diese Anzahl geringer als unendlich, kann man davon ausgehen, das nicht die ganze Charakteristik der Daten in diesen N Datenpunkten erfasst wird. Deswegen sollte man statistische Werte, die aus endlichen Datenmengen entstehen nur als Schätzungen der wahren Statistik betrachten. Aus den endlichen Datenmengen entstehen nun auch wieder die statistischen Schätzungen wie der **Mittelwert** und die **Varianz** deiniert als

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.60}$$

$$S_x^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{2.61}$$

  wobei $x_i - \bar{x}$ als Abweichung von x_i bezeichnet wird. Die **Standardabweichung** S_x wird definiert als $\sqrt{S_x^2}$. Der Mittelwert stellt eine brauchbare Schätzung des wahren Mittelwertes $\bar{x}$ dar. Für eine Normalverteilung von x um einen Mittelwert kann man folgendes statistisch festlegen:

$$x_i = \bar{x} + t_{\nu,P} S_x(P\%) \tag{2.62}$$

  wobei die Variable $t_{\nu,p}$ aus einer neuen gewichtenden Funktion für endliche Datenmengen erhalten wird. Der Wert für die t-Schätzfunktion ist eine Funktion der Wahrscheinlichkeit P und den Freiheitsgraden ν in der Standardabweichung. In N endlichen

Messungen eines gemessenen Wertes, der eine zentrale Tendenz hat, müssen die Daten rund um einen Mittelwert gestreut sein. Die Freiheit jedes Datenpunktes um irgendeinen Wert zu erreichen, wird von diesem Mittelwert eingeschränkt. Infolgedessen ist die Anzahl der Freiheitsgrade ν gleich N-1.

$$\nu = N - 1 \tag{2.63}$$

Die genauen Werte für $t_{\nu,p}$ kann man folgender Tabelle entnehmen:

ν	t_{50}	t_{90}	t_{95}	t_{99}
1	1.000	6.314	12.706	63.657
2	0.816	2.920	4.303	9.925
3	0.765	2.353	3.182	5.841
4	0.741	2.132	2.770	4.604
5	0.727	2.015	2.571	4.032
6	0.718	1.943	2.447	3.707
7	0.711	1.895	2.365	3.499
8	0.706	1.860	2.306	3.355
9	0.703	1.833	2.262	3.250
10	0.700	1.812	2.228	3.169
11	0.697	1.796	2.201	3.106
12	0.695	1.782	2.179	3.055
13	0.694	1.771	2.160	3.012
14	0.692	1.761	2.145	2.977
15	0.691	1.753	2.131	2.947
16	0.690	1.746	2.120	2.921
17	0.689	1.740	2.110	2.898
18	0.688	1.734	2.101	2.878
19	0.688	1.729	2.093	2.861
20	0.687	1.725	2.086	2.845
21	0.686	1.721	2.080	2.831
30	0.683	1.697	2.042	2.750
40	0.681	1.684	2.021	2.704
50	0.680	1.679	2.010	2.679
60	0.679	1.671	2.000	2.660
∞	0.674	1.645	1.960	2.576

Tabelle 2.2: Student t Verteilung

Ich habe nun diese Student t Verteilung für die statistische Ausreisserkorrektur der Konturpunkte verwendet. Damit ist dann die Anpassung der Linien und Kreise genauer, da die Datenmenge genau von diesen Ausreissern befreit wurde.

2.7.3 Beispiel

Als Beispiel wird hier ein 3D-Block dargestellt, bei dem gewisse Teile abgebrochen sind, die man links im Bild erkennen kann. Die statistische Ausreißerkorrektur liefert nun folgendes Ergebnis (Abbildung 2.24 2.25).

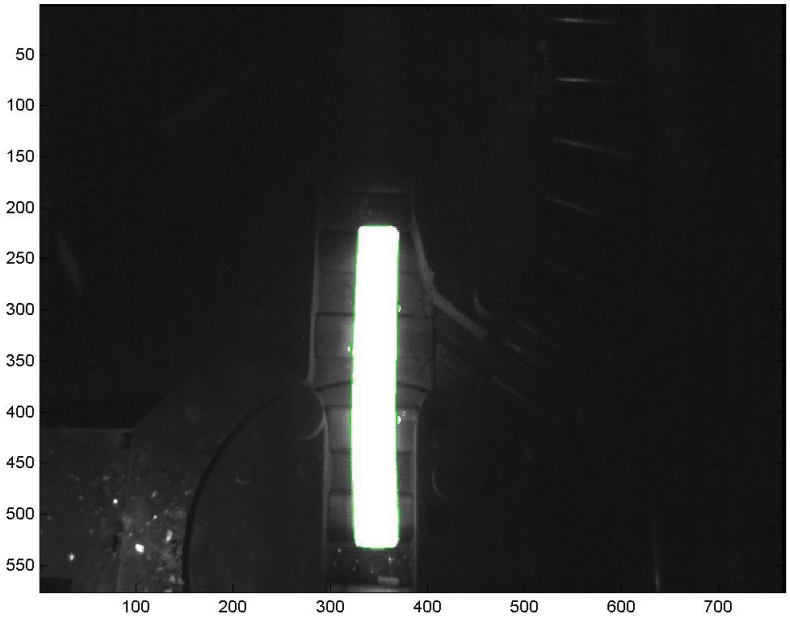


Abbildung 2.24: Statistikbeispiel

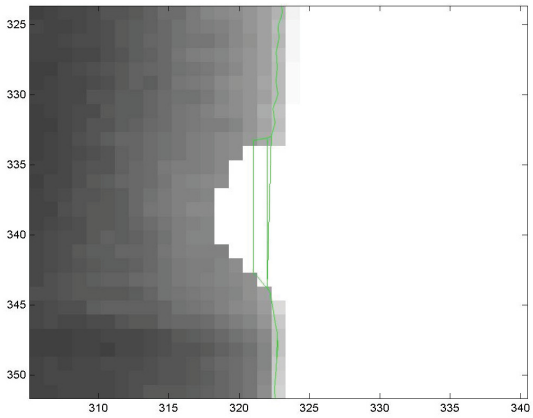


Abbildung 2.25: Statistikbeispiel Detailansicht

Die einzelnen Datenmengen sind im Bild in grün eingezeichnet. Es wurden drei Iterationsschritte durchgeführt, was sehr zuverlässig die Ausreisser eliminert hat.

2.8 Schnittpunktberechnungen in der Ebene

2.8.1 Ziel

Das Ziel besteht darin zu ermitteln, ob sich 2 Kreise oder 2 Linien oder eine Linie mit einem Kreis schneiden und wenn ja zu ermitteln wo die Schnittpunkte liegen.

2.8.2 Theorie, Lösungsweg, verwendete Algorithmen

Schnittpunkte Kreis Kreis

Zuerst stellt sich uns die Frage wie Kreise zueinander liegen können. Dazu eine erklärende Skizze (Abbildung 2.27). Von jedem Kreis ist die Lage der Mittelpunkte und die Radien bekannt. Betrachtet man das Dreiecksgebilde, das sich in Fall 1 ergibt (Abbildung 2.26)

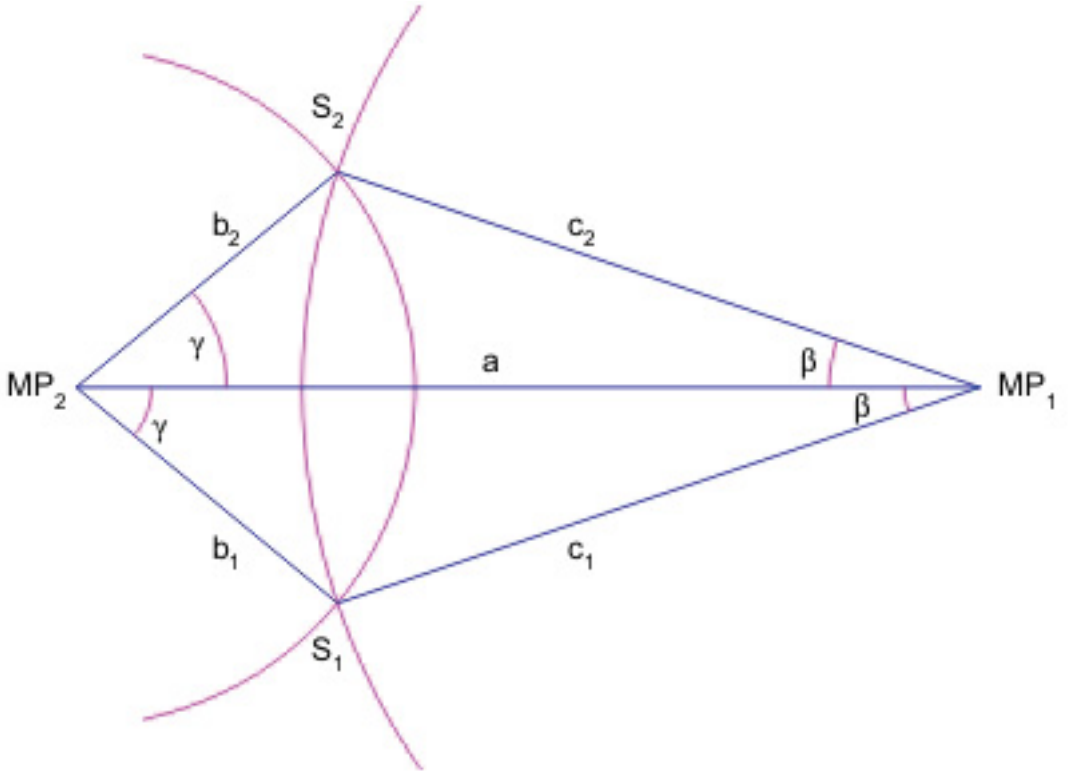


Abbildung 2.26: Schnittpunkte Kreis/Kreis Fall 1

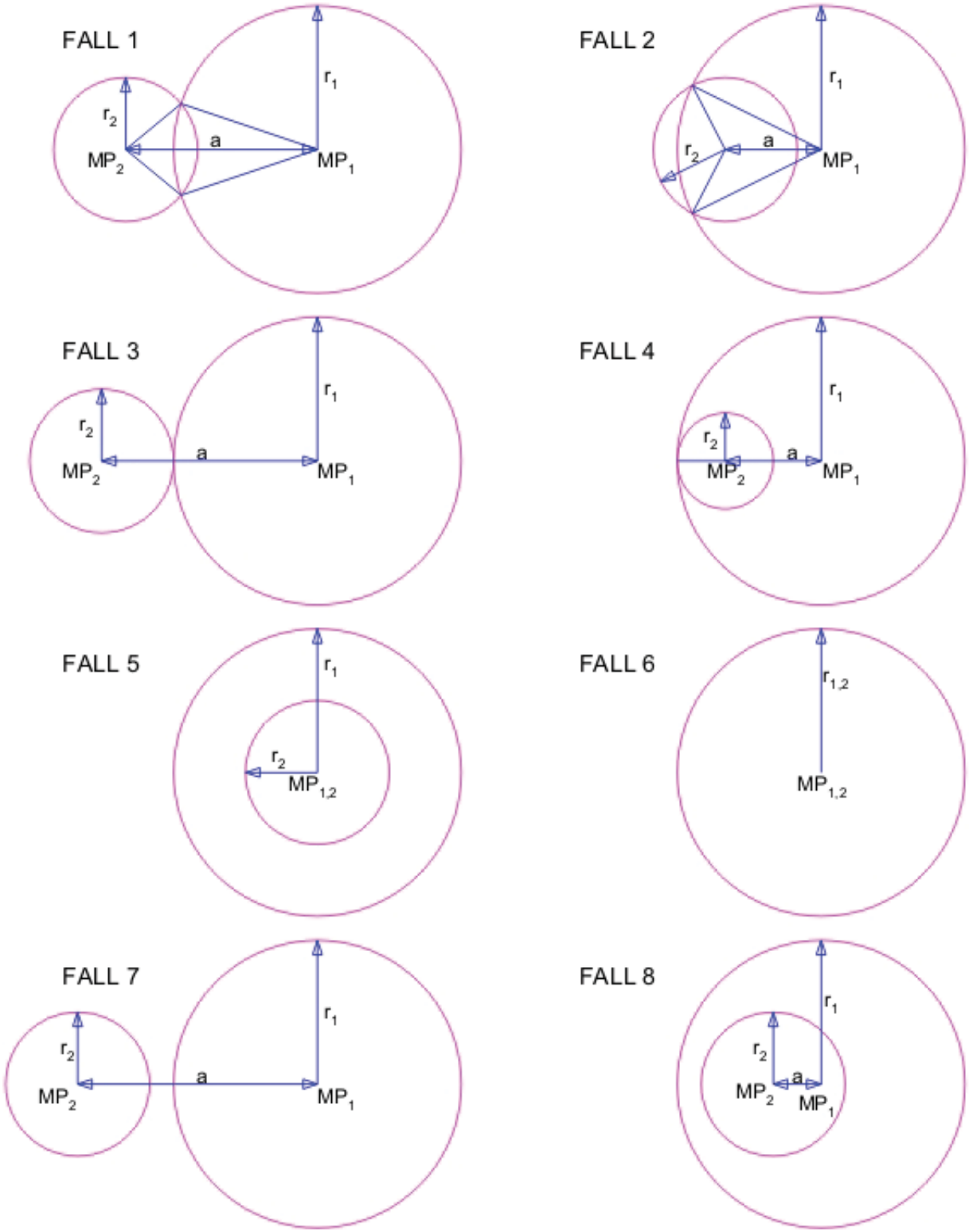


Abbildung 2.27: Schnittpunkte Kreis/Kreis

sieht man 2 schiefwinklige Dreiecke, die die Seite a gemeinsam haben und deren Seite a wie eine Spiegelachse betrachtet werden kann. Bekannt sind alle Seitenlängen des Dreiecks:

$$a = \sqrt{(MPX1 - MPX2)^2 + (MPY1 - MPY2)^2} \tag{2.64}$$

$$b_1 = b_2 = r_2 \tag{2.65}$$

$$c_1 = c_2 = r_1 \tag{2.66}$$

Ermittelt werden soll die Lage der Punkte S_1 und S_2. Diese ist aber völlig unbekannt, da die Lage von b_1, b_2, c_1, c_2 nicht bekannt ist, sondern nur deren Länge. Ohne die Lage jedoch kann man kein Schnittverfahren der Linien durchführen. Bekannt ist uns hingegen die Lage von a, da man die begrenzenden Punkte dieser Linie, sprich die Mittelpunkte der Kreise kennt. Die Lage der Linien b_1, b_2, c_1, c_2 geht nun jeweils durch den Mittelpunkt des Kreises 2 bzw. Kreises 1 und ist jeweils um den Winkel $\pm\gamma$ und $\pm\beta$ vedreht. Die Winkel β und γ kann man nun durch den Cosinussatz bestimmen:

$$b^2 = c^2 + a^2 - 2ca\cos\beta \Rightarrow \beta = \arccos\frac{b^2 - c^2 - a^2}{-2ca} \tag{2.67}$$

und

$$c^2 = a^2 + b^2 - 2ab\cos\gamma \Rightarrow \gamma = \arccos\frac{c^2 - a^2 - b^2}{-2ab} \tag{2.68}$$

aufgrund derselben Drehrichtung, die man festsetzen will, ergibt sich γ zu:

$$\gamma = 360 - \gamma \tag{2.69}$$

Des weiteren werden die Parameter der Gerade a bestimmt, wobei der Nullpunkt im Kreismittelpunkt von Kreis 1 festgelegt wird. Daraus folgt nun:

$$\begin{aligned} x_1 &= 0 \\ y_1 &= 0 \\ x_2 &= MPX2 - MPX1 \\ y_2 &= MPY2 - MPY1 \end{aligned}$$

und damit

$$\begin{aligned} Y_a &= y_1 - y_2 \\ X_a &= x_1 - x_2 \\ N_a &= x_1 y_2 - x_2 y_1 \end{aligned}$$

Die Berechnung der Parameter X, Y, N wurde schon erklärt, es fehlt jedoch eine Erläuterung der

Koordinatentransformation von Linien

Die Erörterungen sollen dadurch vervollständigt werden, dass das Verhalten der Elementargrößen bei Transformationen des rechtwinkligen Koordinatensystems untersucht wird. Das wird dann ein wertvolles Klassifikationsprinzip liefern, durch welches die Grassmannsche Systematik erst ihre feinere Ausführung erhält. Die Formeln der Koordinatenänderung, d.h. die Ausdrücke der Koordinaten x', y' eines Punktes in Bezug auf die neue Lage der Achsen durch die ursprünglichen Koordinaten x, y lauten bekanntlich für die 4 fundamentalen Transformationen des rechtwinkligen Koordinatensystems folgendermaßen:

1. Für die Parallelverschiebung:

$$\begin{aligned} x' &= x + a \\ y' &= y + b \end{aligned} \tag{2.70}$$

2. Für die Drehung um den Winkel φ:

$$\begin{aligned} x' &= x\cos\varphi + y\sin\varphi \\ y' &= -x\sin\varphi + y\cos\varphi \end{aligned} \tag{2.71}$$

3. Für die Spiegelung an der x-Achse:

$$\begin{aligned} x' &= x \\ y' &= -y \end{aligned} \tag{2.72}$$

4. Für die Veränderung des Maßstabes:

$$\begin{aligned} x' &= \lambda x \\ y' &= \lambda y \end{aligned} \tag{2.73}$$

Setzt man Transformationen dieser vier Arten für alle möglichen Werte der Parameter a, b, φ und λ miteinander zusammen, so enstehen die Gleichungen für den allgemeinsten möglichen Übergang von einem rechtwinkligen Koordinatensystem zu einem anderen bei gleichzeitigem Wechsel des Maßstabes. Die Zusammensetzung aller möglichen Verschiebungen und Drehungen entspricht den sämtlichen eigentlichen Bewegungen des Koordinatensystems innerhalb der Ebene. Die Gesamtheit aller dieser Transformationen bildet eine Gruppe, d.h. irgendwelche 2 Transformationen von ihnen zusammengesetzt ergeben wieder eine Transformation

derselben Gesamtheit und die Inverse jeder Transformation ist immer vertreten. Die speziellen Transformationen (Gleichung 2.70 bis 2.73), aus denen man alle anderen zusammensetzen kann, heißen die Erzeugenden der Gruppe.

Bevor man sich überlegt, wie diese einzelnen Transformationen die Determinanten X, Y, N verändern, werden hier noch zwei allgemeine Prinzipien formuliert:

1. Die geometrischen Eigenschaften irgendwelcher Figuren müssen sich stets in Formeln aussprechen lassen, die nicht geändert werden, wenn man das Koordinatensystem abändert, d.h. die Koordinaten sämtlicher Punkte der Figur gleichzeitig einer unserer Transformationen unterwirft und das umgekehrt auch jede Formel, die in diesem Sinn invariant gegen die Gruppe dieser Koordinatentransformationen ist, eine geometrische Eigenschaft darstellen muss . Als einfachste Beispiele, wie sie jeder kennt, sei da nur an den Ausdruck der Entfernung oder des Winkels in der Figur zweier Punkte oder zweier Geraden erinnert. Von diesen und vielen anderen ähnlichen Formeln wird ja im folgenden immer zu handeln sein.

2. Das zweite Prinzip bezieht sich darauf, dass man ein System von analytischen Größen hat, die aus den Koordinaten mehrere Punkte 1,2,... gebildet sind, wie z.B. die drei Größen X, Y, N. Hat dieses System die Eigenschaft, dass es sich bei allen Koordinatentransformationen in bestimmter Weise in sich selbst transformiert, d.h., dass sich das aus den neuen Koordinaten der Punkte 1,2,... in gleicher Weise gebildete Größensystem allein durch die aus den alten Korrdinaten gebildeten Größen ausdrückt (ohne das man die Koordinatenwerte selbst hinzunehmen muss), so sagt man, es definert ein neues geometrisches, d.h. ein vom Koordinatensystem unabhängiges Gebilde.

Das wird nun sogleich an dem von den Grassmannschen Elemantargrößen gelieferten Material näher erläutert. Man unterwirft dazu die beiden Punkte x_1, y_1, x_2, y_2 der gleichen Koordinatentransformation. Beginnt man mit der Parallelverschiebung (Gleichung 2.70) und setzt also:

$$\begin{aligned} x_1' &= x_1 + a, & x_2' &= x_2 + a \\ y_1' &= y_1 + b, & y_2' &= y_2 + b \end{aligned}$$

Vergleicht man die Koordinaten des Linienteils vor und nach der Transformation:

$$\begin{aligned} X &= x_1 - x_2, & Y &= y_1 - y_2, & N &= x_1 y_2 - x_2 y_1, \\ X &= x_1' - x_2', & Y &= y_1' - y_2', & N &= x_1' y_2' - x_2' y_1', \end{aligned}$$

so ergibt sich unmittelbar

1. für die Parallelverschiebung:

$$\begin{aligned} X' &= X \\ Y' &= Y \\ N' &= N + bX - aY \end{aligned} \tag{2.74}$$

2. für die Drehung (Gleichung 2.59):

$$\begin{aligned} X' &= X\cos\varphi + Y\sin\varphi \\ Y' &= -X\sin\varphi + Y\cos\varphi \\ N' &= N \end{aligned} \tag{2.75}$$

3. für die Spiegelung (Gleichung 2.60):

$$\begin{aligned} X' &= X \\ Y' &= -Y \\ N' &= -N \end{aligned} \tag{2.76}$$

4. für die Maßstabsänderung (Gleichung 2.61):

$$\begin{aligned} X' &= \lambda X \\ Y' &= \lambda Y \\ N' &= \lambda^2 N \end{aligned} \tag{2.77}$$

In den letzten Formeln (Gleichung 2.77) tritt ein Unterschied in dem Verhalten der einzelnen Größen auf, da der Exponent der Potenz von λ, mit der sie multipliziert werden, nicht überall derselbe ist. Diesem Unterschied wird man in der Physik durch die Einführung des Dimensionsbegriffes gerecht. X, Y haben die Dimension 1 einer Linie und N hat die Dimension 2 eines Flächeninhaltes.

Doch nun zurück zu dem konkreten Problem. Man muss jetzt die Determinaten X, Y, N der einzelnen Linien bestimmen.

- Die Bestimmung von b_1 erfolgt durch Drehung der Determinanten X_a, Y_a und N_a um $+\gamma$ und Rücktransformation auf den Mittelpunkt des Kreises 2.
- Die Bestimmung von b_2 erfolgt durch Drehung der Determinanten X_a, Y_a und N_a um -γ und Rücktransformation auf den Mittelpunkt des Kreises 2.
- Die Bestimmung von c_1 erfolgt durch Drehung der Determinanten X_a, Y_a und N_a um $+\beta$ und Rücktransformation auf den Mittelpunkt des Kreises 1.

- Die Bestimmung von c_2 erfolgt durch Drehung der Determinanten X_a, Y_a und N_a um -β und Rücktransformation auf den Mittelpunkt des Kreises 1.

Sind nun die Linien b_1, b_2, c_1, c_2 bestimmt, können die Schnittpunkte S_1 und S_2 ermittelt werden, was unmittelbar zu folgendem Thema führt:

Schnittpunkt Linie Linie

Eine Linie kann man im Kapitel 2.6.2 beschrieben durch zwei Punkte definiert werden. Genauso kann aber auch umgekehrt ein Punkt durch zwei Linien definiert werden, indem die Matrix folgendermaen angeschrieben wird:

$$\begin{bmatrix} Y_1 & X_1 & N_1 \\ Y_2 & X_2 & N_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.78}$$

Daraus folgt nun

$$x = \frac{N_2X_1 - N_1X_2}{Y_1X_2 - Y_2X_1} \tag{2.79}$$

und

$$y = \frac{N_2Y_1 - N_1Y_2}{Y_1X_2 - Y_2X_1} \tag{2.80}$$

Somit können die Schnittpunkte S_1 und S_2 eindeutig bestimmt werden!

Mit diesem Verfahren kann man für die Fälle 1,2,3 und 4 die Schnittpunkte bzw. den Schnittpunkt (Im Fall 3 und 4 fällt S_1 und S_2 zusammen) bestimmen. Alle anderen ergeben keine sinnvollen Schnittpunkte.
Daraus ergeben sich folgende Erkenntnisse:

Wert von a	**Fall x**
$a = 0$	Die Kreise haben identische Mittelpunkte. ⇒ Fall 5 und 6.
$a > r_1 + r_2$	Es können keine Schnittpunkte gefunden werden. ⇒ Fall 7
$a < r_1 - r_2$	Es können keine Schnittpunkte gefunden werden. ⇒ Fall 8
Alle anderen Werte von a	Es können keine Schnittpunkte gefunden werden. Ergeben einen der ersten 4 Fälle und die Schnittpunkte können mit obiger Berechnung ermittelt werden.

Tabelle 2.3: Schnittpunktfälle Kreis Kreis

Schnittpunkt Linie Kreis

Betrachtet man nun das Problem der Schnittpunktfindung Linie Kreis muss zuerst einmal ein Kriterium gefunden werden, dass eine Auskunft darüber liefert, ob der Kreis überhaupt geschnitten wird. Diese Auskunft liefert der Normalabstand der Linie zum Kreismittelpunkt und er kann wie folgt ermittelt werden:

1. Verschiebung der Linie 1 (das ist die Linie die mit dem Kreis geschnitten werden soll) in den Nullpunkt: Wie man jetzt schon weiß, bleiben die Parameter X und Y dieselben und N hat den Wert 0 im Nullpunkt.

2. Drehung der Linie um 90:

$$\begin{aligned} \cos(90) &= 0 \\ \sin(90) &= 1 \\ X_{neu} &= Y \\ Y_{neu} &= -X \\ N_{neu} &= 0 \end{aligned}$$

3. Verschiebung der neuen Linie in den Kreismittelpunkt.

$$\begin{aligned} X_{neu1} &= X_{neu} \\ Y_{neu1} &= Y_{neu} \\ N_{neu1} &= N_{neu} + y_{Kreis}X_{neu1} - x_{Kreis} * Yneu1 \end{aligned}$$

4. Schnittpunktermittlung dieser Linie mit unserer Linie 1: Nach Umformen der Gleichung lautet das Ergebnis:

$$\begin{aligned} x &= \frac{X(y_{Kreis}Y + x_{Kreis}X) - NY}{X^2 + Y^2} \\ y &= \frac{Y(y_{Kreis}Y + x_{Kreis}X) + NX}{X^2 + Y^2} \end{aligned}$$

5. Das Entscheidungskriterium lautet nun:

$$Normalabstand = \sqrt{(x - x_{Kreis})^2 + (y - y_{Kreis})^2} \tag{2.81}$$

Daraus ergeben sich folgende Erkenntnisse:

Wert des Normalabstandes	**Erkenntnis**
$Normalabstand > r_{Kreis}$	Die Linie schneidet den Kreis nicht.
$Normalabstand = r_{Kreis}$	Die Linie bildet eine Tangente zum Kreis und es gibt einen Schnittpunkt mit den Koordinaten x und y wie oben berechnet.
$Normalabstand < r_{Kreis}$	Die Linie schneidet den Kreis in 2 Punkten

Tabelle 2.4: Schnittpunktfälle Linie Kreis

Im Fall zweier Schnittpunkte können diese wie folgt bestimmt werden:

1. Bestimmung des Winkels α:

$$\alpha = \arccos \frac{Normalabstand}{r_{Kreis}}$$

2. Drehung der Linie aus obigen Punkt 2 um $\pm\alpha$.

3. Verschieben dieser 2 neu gewonnenen Linien in den Kreismittelpunkt.

4. Schneiden der 2 Linien mit unserer Linie 1. Daraus ergibt sich nun die Endformel:

$$\begin{aligned} x_{1,2} &= x + Xtan(\pm\alpha)\frac{N + x_{Kreis}Y - y_{Kreis}X}{X^2 + Y^2} \\ y_{1,2} &= y + Ytan(\pm\alpha)\frac{N + x_{Kreis}Y - y_{Kreis}X}{X^2 + Y^2} \end{aligned}$$

Das besondere an diesem Schnittverfahren ist, dass alle Sonderfälle (die Linie hat die gleiche Lage wie die Achsen oder geht durch den Nullpunkt) berücksichtigt werden und nie eine Division durch 0 ensteht!

2.8.3 Beispiel

Als Beispiel wird hier ein 3D-Block dargestellt, bei dem gewisse Teile abgebrochen sind, die man links im Bild erkennen kann. Die Schnittpunktbestimmung liefert nun folgendes Ergebnis (Abbildung 2.24).

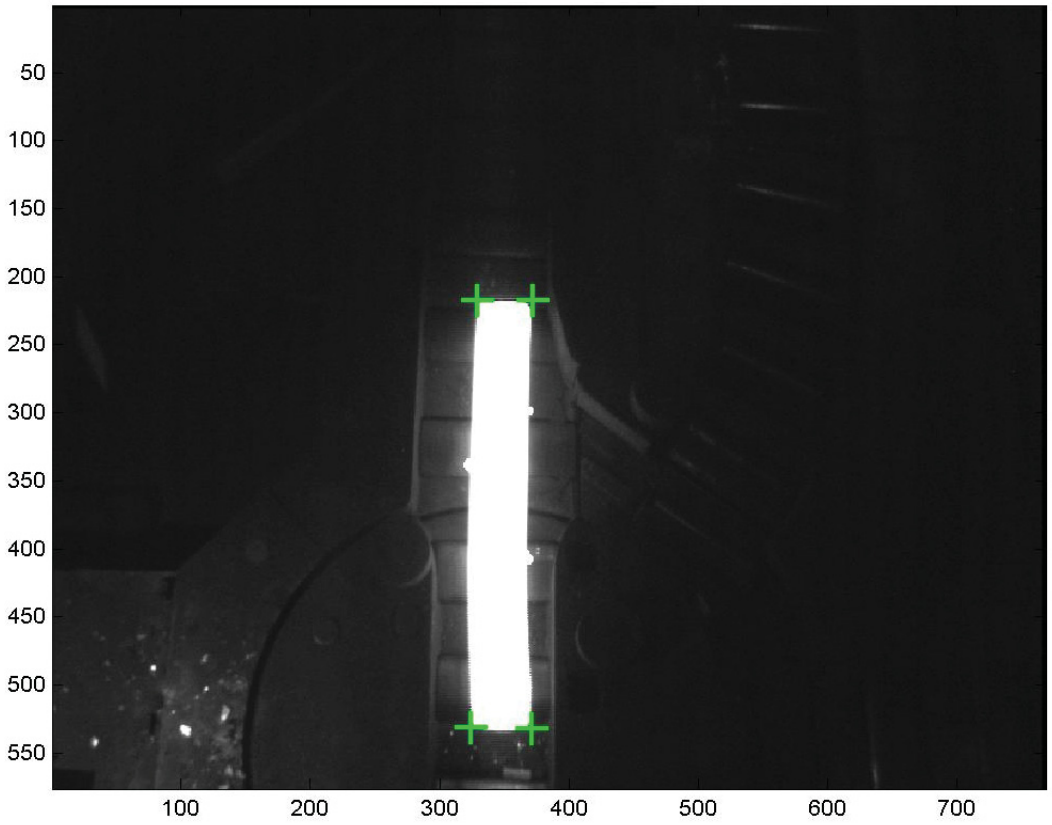


Abbildung 2.28: Ermittelte Schnittpunkte

Die einzelnen Schnittpunkte sind im Bild in grün eingezeichnet. Nach diesem Verfahren besitzt man die Information über die Lage genau definierter Eckpunkte und die angeglichene Kurve (Kreis oder Linie), die zwischen den Eckpunkten liegt.

Kapitel 3

Quellcode

3.1 Konturberechnung

```
function mycontour=GContour(Imagematrix,Treshold,Linespec,fig)
%
% Use:  mycontour=GContour(Imagematrix,Treshold,Linespec,fig);
%   or  mycontour=GContour(Imagematrix,Treshold);
%
% Description: This routine calculates a contour at a specified
%              threshold for a matrix of an image.
%
% Input Parameters:
%   Imagematrix: The matrix of the image
%   Treshold:    The threshold value.
%   Linespec:    The color- and linespecification of the contours
%                in the figure. (optional)
%   fig:         Handle to the figure on which the plot is to be made.
%               (optional)
%
% Return Parameters:
%   mycontour:   The structure array.
%
% By:  Georg Egger
% Date: 10.Dec.2001
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
```

```
% Date:          Comment:
% 10.12.2001     Original Version 1.0
%------------------------------------------------------------------------
I1=Imagematrix;
Schwelle=Treshold;
[ze sp]= size(I1);
Kn=0;
koor=[0 0];
koorex=[0 0 0 0];
counter=0;
counter1=0;
for x=1:sp
 for y=1:ze
  %If the actual value is higher than the threshold the sorrounding area.
  %is started to be checked
  if I1(y,x)>Schwelle
   i=y;
   j=x;
   Richtung(1:4)=0;
   if j<sp & I1(i,j+1)<=Schwelle Richtung(1)=1; end;
   if i>1 & I1(i-1,j)<=Schwelle  Richtung(2)=1; end;
   if j>1 & I1(i,j-1)<=Schwelle  Richtung(3)=1; end;
   if i<ze & I1(i+1,j)<=Schwelle Richtung(4)=1; end;
  end;
  if I1(y,x)>Schwelle & sum(Richtung)>0
   %Calculation if the point was already proved.
   untersucht=0;
   [ze1 sp1]=size(koor);
   for c=1:ze1
    if koor(c,1)==j & koor(c,2)==i untersucht=1; end;
   end;
   if untersucht==1
   else
    if sum(Richtung)==0
    else
     %Beginning with the calculation of the contour.
     koorex1=[0 0];
     koordel=[0 0];
     Kn=Kn+1;
     fertig=0;
     Startpunktx=j;
     Startpunkty=i;
     startx=0;
     starty=0;
     Konturx(1,Kn)=0;
```

```
     Kontury(1,Kn)=0;
     Ln=0;
     grenze=0;
     rv=[0 0 0 0];
     rv1=[0 0 0 0];
     anfang=counter;
     Richt=0;
     while fertig==0
       Sb=0;
       SW=Schwelle;
       AW=I1(i,j);
       % The information about the number and direction of contour
       % points around the actual position is saved in the parameter
       % Richtung! Dependent on Richtung the contour points are
       % calculated.

       %--------------------------------------------------------------------
       % The explanation of the used functions are written in the
       % functions themselves.
       %--------------------------------------------------------------------
%-------------------------------------------------------------------------
       if Richtung==[1 0 0 0] | (Richtung==[0 0 0 0] & rv==[1 0 0 0])
        if Richtung==[0 0 0 0]
         rv=[0 0 0 0];
         [i,j,grenze,fertig]=search(1,i,j,grenze,sp,ze,fertig);
        else
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,4,sp,ze,...
             grenze,startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),...
          Kontury(Ln,Kn),counter,koor(counter,:)]=...
          MultiPointCal(0,I1(i,j+1),SW,AW,1,i,j,Ln,grenze,sp,ze,...
          counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
          counter,koor(counter,:)]=...
          MultiPointCal(0,I1(i,j+1),SW,AW,1,i,j,Ln,grenze,sp,ze,...
          counter,fertig,koor(anfang+1,:),...
          Konturx(Ln,Kn),Kontury(Ln,Kn));
         end;
        end;
       end;
%-------------------------------------------------------------------------
       if Richtung==[0 1 0 0] | (rv==[0 1 0 0] & Richtung==[0 0 0 0])
        if Richtung==[0 0 0 0]
```

```
         rv=[0 0 0 0];
         [i,j,grenze,fertig]=search(2,i,j,grenze,sp,ze,fertig);
        else
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,1,sp,ze,...
             grenze,startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),...
          Kontury(Ln,Kn),counter,koor(counter,:)]=...
          MultiPointCal(0,I1(i-1,j),SW,AW,2,i,j,Ln,grenze,sp,...
          ze,counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
          counter,koor(counter,:)]=...
          MultiPointCal(0,I1(i-1,j),SW,AW,2,i,j,Ln,grenze,sp,ze,...
          counter,fertig,koor(anfang+1,:),...
          Konturx(Ln,Kn),Kontury(Ln,Kn));
         end;
        end;
       end;
%-------------------------------------------------------------------------
       if Richtung==[0 0 1 0] | (rv==[0 0 1 0] & Richtung==[0 0 0 0])
        if Richtung==[0 0 0 0]
         rv=[0 0 0 0];
         [i,j,grenze,fertig]=search(3,i,j,grenze,sp,ze,fertig);
        else
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,2,sp,ze,...
             grenze,startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),...
          Kontury(Ln,Kn),counter,koor(counter,:)]=...
          MultiPointCal(0,I1(i,j-1),SW,AW,3,i,j,Ln,grenze,sp,...
          ze,counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
          counter,koor(counter,:)]=...
          MultiPointCal(0,I1(i,j-1),SW,AW,3,i,j,Ln,grenze,sp,ze,...
          counter,fertig,koor(anfang+1,:),...
          Konturx(Ln,Kn),Kontury(Ln,Kn));
         end;
        end;
       end;
%-------------------------------------------------------------------------
       if Richtung==[0 0 0 1] | (rv==[0 0 0 1] & Richtung==[0 0 0 0])
        if Richtung==[0 0 0 0]
         rv=[0 0 0 0];
```

```
          [i,j,grenze,fertig]=search(4,i,j,grenze,sp,ze,fertig);
         else
          [startx,starty,grenze,rv1]=...
              Startpoint(Startpunkty,Startpunktx,i,j,rv1,3,sp,ze,...
              grenze,startx,starty);
          if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),...
           Kontury(Ln,Kn),counter,koor(counter,:)]=...
           MultiPointCal(0,I1(i+1,j),SW,AW,4,i,j,Ln,grenze,sp,...
           ze,counter,fertig);
          else
           [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
           counter,koor(counter,:)]=...
           MultiPointCal(0,I1(i+1,j),SW,AW,4,i,j,Ln,grenze,sp,ze,...
           counter,fertig,koor(anfang+1,:),...
           Konturx(Ln,Kn),Kontury(Ln,Kn));
          end;
         end;
        end;
%-------------------------------------------------------------------------
        if Richtung==[1 1 0 0]
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,4,sp,ze,...
             grenze,startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
           Kontury(Ln-dif:Ln,Kn),counter,koor(counter,:)]=...
           MultiPointCal(0,[I1(i,j+1) I1(i-1,j)],...
           SW,AW,[1 2],i,j,Ln,grenze,sp,ze,counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=MultiPointCal(0,[I1(i,j+1) I1(i-1,j)],SW,...
          AW,[1 2],i,j,Ln,grenze,...
          sp,ze,counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),...
          Kontury(Ln,Kn));
         end;
        end;
%-------------------------------------------------------------------------
        if Richtung==[1 0 1 0]
         %Special case
         %-----------------------------------------------
         %The explanation is valid for all special cases!!
         %-----------------------------------------------
         %There are exactly 6 possibilities:
         %1. case:
         %The special case is invoked the first time and there are
```

```
%already  several contour points calculated in the actual
%contour and the actual contour will be closed with invoking
%this coordinate in the actual contour once again.-> The actual
%coordinate has not to be proved again.
%2. case:
%The special case is invoked the first time and there are
%already several contour points calculated in the actual
%contour and the actual contour will be closed without invoking
%this coordinate in the actual contour once again.-> The actual
%coordinate has to be proved once again.
%3. case:
%The special case is invoked the first time and there are
%already several contour points calculated in the actual
%contour and the actual contour will be closed without invoking
%this coordinate in the actual contour once again.-> The actual
%coordinate has to be proved once again.
%4. case:
%The special case is invoked the first time and the actual
%coordinate is the startpoint of the contour and the actual
%contour will be closed with invoking this coordinate in the
%actual contour two times again.-> The actual coordinate has not
%to be proved once again.
%5. case:
%The special case is invoked another time at the actual
%coordinate and it is the startpoint of a contour so the actual
%contour will be closed with invoking this coordinate once
%again.->The actual position was scaned in case 1 combined with
%case 5 four times and in case 3 combined with case 5 three times.
%6. case:
%The special case is invoked another time at the actual
%coordinate and it is not the startpoint of a contour so the
%actual contour will be closed without invoking this coordinate
%once again.->The actual position was scaned in case 1 combined
%with case 6 three times and in case 3 combined with case 6 two
%times.
xy=[j,i];
hit=0;
Richt=0;
hit1=0;
delpoint=0;
[zelr splr]=size(koorex);
for h=1:zelr
    if xy==koorex(h,1:2)
        hit=1;
        hit1=h;
```

```
    end;
end;
switch Ln
case 0
 switch hit
 case 0
     counter1=counter1+1;
     Richt=3;
     koorex(counter1,:)=[j i Richt Kn];
 case 1
     if koorex(hit1,3)==3 Richt=1; end;
     if koorex(hit1,3)==1 Richt=3; end;
     delpoint=1;
 end;
 if Richt==1 [startx,starty,grenze,rv1]=...
         Startpoint(Startpunkty,Startpunktx,i,j,rv1,4,sp,ze,...
             grenze,startx,starty); end;
 if Richt==3 [startx,starty,grenze,rv1]=...
         Startpoint(Startpunkty,Startpunktx,i,j,rv1,2,sp,ze,...
             grenze,startx,starty); end;
otherwise
 if Konturx(Ln,Kn)>j Richt=1; end;
 if Konturx(Ln,Kn)<=j Richt=3; end;
 switch hit
 case 0
  counter1=counter1+1;
  koorex(counter1,:)=[j i Richt Kn];
 case 1
  if koorex(hit1,4)==Kn
   if Richt==koorex(hit1,3)
   else
    delpoint=1;
   end;
  else
   delpoint=1;
  end;
 end;
end;
if Richt==1 Input=I1(i,j+1); end;
if Richt==3 Input=I1(i,j-1); end;
if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),...
            Kontury(Ln,Kn),counter,koor(counter,:)]=...
  MultiPointCal(0,Input,SW,AW,Richt,i,j,Ln,grenze,sp,ze,...
  counter,fertig);
else
```

```
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
          counter,koor(counter,:)]=...
          MultiPointCal(0,Input,SW,AW,Richt,i,j,Ln,grenze,sp,ze,counter,...
          fertig,koor(anfang+1,:),...
          Konturx(Ln,Kn),Kontury(Ln,Kn));
         end;
         if delpoint==1  koordel(counter-anfang,1:2)=xy;
         else
          koordel(counter-anfang,1:2)=[0 0];
         end;
         koorex1(counter-anfang,1:2)=xy;
        end;
%-------------------------------------------------------------------------
        if Richtung==[1 0 0 1]
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,3,sp,ze,grenze,...
             startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
           Kontury(Ln-dif:Ln,Kn),counter,...
           koor(counter,:)]=MultiPointCal(0,[I1(i+1,j) I1(i,j+1)],...
           SW,AW,[4 1],i,j,Ln,grenze,sp,ze,counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=MultiPointCal(0,[I1(i+1,j) I1(i,j+1)],SW,...
          AW,[4 1],i,j,Ln,grenze,sp,ze,counter...
          ,fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
         end;
        end;
%-------------------------------------------------------------------------
        if Richtung==[1 1 1 0]
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,4,sp,ze,grenze,...
             startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=MultiPointCal(0,...
          [I1(i,j+1) I1(i-1,j) I1(i,j-1)],SW,AW,[1 2 3],i,j,Ln,...
          grenze,sp,ze,counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=...
          MultiPointCal(0,[I1(i,j+1) I1(i-1,j) I1(i,j-1)],...
          SW,AW,[1 2 3],i,j,Ln,grenze,...
```

```
          sp,ze,counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),...
          Kontury(Ln,Kn));
         end;
        end;
%-------------------------------------------------------------------------
        if Richtung==[1 1 0 1]
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,3,sp,ze,grenze,...
             startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=MultiPointCal(0,...
          [I1(i+1,j) I1(i,j+1) I1(i-1,j)],SW,AW,[4 1 2],i,j,Ln,...
          grenze,sp,ze,counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=...
          MultiPointCal(0,[I1(i+1,j) I1(i,j+1) I1(i-1,j)],...
          SW,AW,[4 1 2],i,j,Ln,grenze,...
          sp,ze,counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),...
          Kontury(Ln,Kn));
         end;
        end;
%-------------------------------------------------------------------------
        if Richtung==[1 0 1 1]
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,2,sp,ze,grenze,...
             startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=MultiPointCal(0,...
          [I1(i,j-1) I1(i+1,j) I1(i,j+1)],SW,AW,[3 4 1],i,j,Ln,...
          grenze,sp,ze,counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=...
          MultiPointCal(0,[I1(i,j-1) I1(i+1,j) I1(i,j+1)],...
          SW,AW,[3 4 1],i,j,Ln,grenze,...
          sp,ze,counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),...
          Kontury(Ln,Kn));
         end;
        end;
%-------------------------------------------------------------------------
```

```
        if Richtung==[1 1 1 1]
         [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
         Kontury(Ln-dif:Ln,Kn),counter,...
         koor(counter,:)]=MultiPointCal(0,...
         [I1(i,j+1) I1(i-1,j) I1(i,j-1) I1(i+1,j)],SW,AW,[1 2 3 4],i,j,...
         Ln,grenze,sp,ze,counter,fertig);
        end;
%-------------------------------------------------------------------------
        if Richtung==[0 1 1 0]
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,1,sp,ze,grenze,...
             startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
           Kontury(Ln-dif:Ln,Kn),counter,...
           koor(counter,:)]=MultiPointCal(0,[I1(i-1,j) I1(i,j-1)],...
           SW,AW,[2 3],i,j,Ln,grenze,...
           sp,ze,counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=MultiPointCal(0,[I1(i-1,j) I1(i,j-1)],SW,...
          AW,[2 3],i,j,Ln,grenze,sp,ze,counter...
          ,fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
         end;
        end;
%-------------------------------------------------------------------------
        if Richtung==[0 1 0 1]
         %special case
         xy=[j,i];
         hit=0;
         Richt=0;
         hit1=0;
         delpoint=0;
         [zelr splr]=size(koorex);
         for h=1:zelr
          if xy==koorex(h,1:2)
           hit=1;
           hit1=h;
          end;
         end;
         switch Ln
         case 0
          switch hit
           case 0
            counter1=counter1+1;
```

```
   Richt=2;
   koorex(counter1,:)=[j i Richt Kn];
  case 1
   if koorex(hit1,3)==2 Richt=4; end;
   if koorex(hit1,3)==4 Richt=2; end;
   delpoint=1;
 end;
 if Richt==2 [startx,starty,grenze,rv1]=...
        Startpoint(Startpunkty,Startpunktx,i,j,rv1,1,sp,ze,...
            grenze,startx,starty); end;
 if Richt==4 [startx,starty,grenze,rv1]=...
        Startpoint(Startpunkty,Startpunktx,i,j,rv1,3,sp,ze,...
            grenze,startx,starty); end;
otherwise
 if Kontury(Ln,Kn)>i Richt=4; end;
 if Kontury(Ln,Kn)<=i Richt=2; end;
 switch hit
  case 0
   counter1=counter1+1;
   koorex(counter1,:)=[j i Richt Kn];
  case 1
   if koorex(hit1,4)==Kn
    if Richt==koorex(hit1,3)
    else
     delpoint=1;
    end;
   else
    delpoint=1;
   end;
 end;
end;
if Richt==2 Input=I1(i-1,j); end;
if Richt==4 Input=I1(i+1,j); end;
if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),...
        Kontury(Ln,Kn),counter,koor(counter,:)]=...
        MultiPointCal(0,Input,SW,AW,Richt,i,j,Ln,grenze,sp,...
        ze,counter,fertig);
else
 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
        counter,koor(counter,:)]=...
 MultiPointCal(0,Input,SW,AW,Richt,i,j,Ln,grenze,sp,ze,...
 counter,fertig,koor(anfang+1,:),...
 Konturx(Ln,Kn),Kontury(Ln,Kn));
end;
if delpoint==1 koordel(counter-anfang,1:2)=xy;
```

```
         else
          koordel(counter-anfang,1:2)=[0 0];
         end;
         koorex1(counter-anfang,1:2)=xy;
        end;
%-------------------------------------------------------------------------
        if Richtung==[0 1 1 1]
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,1,sp,ze,grenze,...
             startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
           Kontury(Ln-dif:Ln,Kn),counter,koor(counter,:)]=...
           MultiPointCal(0,[I1(i-1,j) I1(i,j-1) I1(i+1,j)],SW,AW,...
           [2 3 4],i,j,Ln,grenze,sp,ze,counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,koor(counter,:)]=MultiPointCal(0,...
          [I1(i-1,j) I1(i,j-1) I1(i+1,j)],SW,AW,[2 3 4],i,j,Ln,grenze,...
          sp,ze,counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),...
          Kontury(Ln,Kn));
         end;
        end;
%-------------------------------------------------------------------------
        if Richtung==[0 0 1 1]
         [startx,starty,grenze,rv1]=...
             Startpoint(Startpunkty,Startpunktx,i,j,rv1,2,sp,ze,grenze,...
             startx,starty);
         if Ln==0 [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=MultiPointCal(0,[I1(i,j-1) I1(i+1,j)],...
          SW,AW,[3 4],i,j,Ln,grenze,sp,...
          ze,counter,fertig);
         else
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
          Kontury(Ln-dif:Ln,Kn),counter,...
          koor(counter,:)]=MultiPointCal(0,[I1(i,j-1) I1(i+1,j)],SW,...
          AW,[3 4],i,j,Ln,grenze,sp,ze,...
          counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
         end;
        end;
%-------------------------------------------------------------------------
        if fertig==0
            % The information about the number and direction of contour
            % points around the actual position is saved in the parameter
            % Richtung!
```

```
            Richtung(1:4)=0;
            if j<sp & I1(i,j+1)<=Schwelle Richtung(1)=1; end;
            if i>1 & I1(i-1,j)<=Schwelle Richtung(2)=1; end;
            if j>1 & I1(i,j-1)<=Schwelle Richtung(3)=1; end;
            if i<ze & I1(i+1,j)<=Schwelle Richtung(4)=1; end;
            if rv==[1 0 0 0] & Richtung(1)==0 & sum(Richtung)>0 ...
                    Richtung=[0 0 0 0]; end;
            if rv==[0 0 0 1] & Richtung(4)==0 & sum(Richtung)>0 ...
                    Richtung=[0 0 0 0]; end;
            if rv==[0 0 1 0] & Richtung(3)==0 & sum(Richtung)>0 ...
                    Richtung=[0 0 0 0]; end;
            if rv==[0 1 0 0] & Richtung(2)==0 & sum(Richtung)>0 ...
                    Richtung=[0 0 0 0]; end;
        end;
%-------------------------------------------------------------------------
     end; %while
     gr=grenze;
     if sum(grenze)==1
      %The chronological order of the vector has to be turned, because
      %the route of scaning the contour is turned too!
      for u=1:Ln
       Kontx(u,Kn)=Konturx(Ln+1-u,Kn);
       Konty(u,Kn)=Kontury(Ln+1-u,Kn);
      end;
      Konturx(1:Ln,Kn)=Kontx(1:Ln,Kn);
      Kontury(1:Ln,Kn)=Konty(1:Ln,Kn);
      i=starty;
      j=startx;
      rv=rv1;
      % The information about the number and direction of contour points
      % around the actual position is saved in the parameter Richtung!
      Richtung(1:4)=0;
      if j<sp & I1(i,j+1)<=Schwelle Richtung(1)=1; end;
      if i>1 & I1(i-1,j)<=Schwelle Richtung(2)=1; end;
      if j>1 & I1(i,j-1)<=Schwelle Richtung(3)=1; end;
      if i<ze & I1(i+1,j)<=Schwelle Richtung(4)=1; end;
      if rv==[1 0 0 0] & Richtung(1)==0 & sum(Richtung)>0 ...
              Richtung=[0 0 0 0]; end;
      if rv==[0 0 0 1] & Richtung(4)==0 & sum(Richtung)>0 ...
              Richtung=[0 0 0 0]; end;
      if rv==[0 0 1 0] & Richtung(3)==0 & sum(Richtung)>0 ...
              Richtung=[0 0 0 0]; end;
      if rv==[0 1 0 0] & Richtung(2)==0 & sum(Richtung)>0 ...
              Richtung=[0 0 0 0]; end;
      fertig=0;
```

```
      while fertig==0
       SW=Schwelle;
       AW=I1(i,j);
%-------------------------------------------------------------------------
       if Richtung==[1 0 0 0] | (Richtung==[0 0 0 0] & rv==[1 0 0 0])
         if Richtung==[0 0 0 0]
           rv=[0 0 0 0];
           [i,j,grenze,fertig]=search(1,i,j,grenze,sp,ze,fertig);
         else
           [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
           counter,koor(counter,:)]=...
           MultiPointCal(1,I1(i,j+1),SW,AW,1,i,j,Ln,grenze,sp,ze,...
           counter,fertig,koor(anfang+1,:),...
           Konturx(Ln,Kn),Kontury(Ln,Kn));
         end;
       end;
%-------------------------------------------------------------------------
       if Richtung==[0 1 0 0] | (rv==[0 1 0 0] & Richtung==[0 0 0 0])
          if Richtung==[0 0 0 0]
           rv=[0 0 0 0];
           [i,j,grenze,fertig]=search(2,i,j,grenze,sp,ze,fertig);
          else
           [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
           counter,koor(counter,:)]=...
           MultiPointCal(1,I1(i-1,j),SW,AW,2,i,j,Ln,grenze,sp,ze,...
           counter,fertig,koor(anfang+1,:),...
           Konturx(Ln,Kn),Kontury(Ln,Kn));
          end;
       end;
%-------------------------------------------------------------------------
       if Richtung==[0 0 1 0] | (rv==[0 0 1 0] & Richtung==[0 0 0 0])
          if Richtung==[0 0 0 0]
          rv=[0 0 0 0];
          [i,j,grenze,fertig]=search(3,i,j,grenze,sp,ze,fertig);
          else
           [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
           counter,koor(counter,:)]=...
           MultiPointCal(1,I1(i,j-1),SW,AW,3,i,j,Ln,grenze,sp,ze,...
           counter,fertig,koor(anfang+1,:),...
           Konturx(Ln,Kn),Kontury(Ln,Kn));
          end;
       end;
%-------------------------------------------------------------------------
       if Richtung==[0 0 0 1] | (rv==[0 0 0 1] & Richtung==[0 0 0 0])
         if Richtung==[0 0 0 0]
```

```
          rv=[0 0 0 0];
          [i,j,grenze,fertig]=search(4,i,j,grenze,sp,ze,fertig);
        else
           [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
           counter,koor(counter,:)]=...
           MultiPointCal(1,I1(i+1,j),SW,AW,4,i,j,Ln,grenze,sp,ze,...
           counter,fertig,koor(anfang+1,:),...
           Konturx(Ln,Kn),Kontury(Ln,Kn));
        end;
      end;
%-------------------------------------------------------------------------
      if Richtung==[1 1 0 0]
       [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
       Kontury(Ln-dif:Ln,Kn),counter,...
       koor(counter,:)]=MultiPointCal(1,[I1(i-1,j) I1(i,j+1)],SW,AW,...
       [2 1],i,j,Ln,grenze,sp,ze,...
       counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
      end;
%-------------------------------------------------------------------------
      if Richtung==[1 0 1 0]
          %Special case
          xy=[j,i];
          hit=0;
          Richt=0;
          hit1=0;
          delpoint=0;
          [zelr splr]=size(koorex);
          for h=1:zelr
              if xy==koorex(h,1:2)
                  hit=1;
                  hit1=h;
              end;
          end;
          if Konturx(Ln,Kn)>j Richt=1; end;
          if Konturx(Ln,Kn)<=j Richt=3; end;
          switch hit
          case 0
             counter1=counter1+1;
             koorex(counter1,:)=[j i Richt Kn];
          case 1
             if koorex(hit1,4)==Kn
               if Richt==koorex(hit1,3)
               else
                  delpoint=1;
               end;
```

```
            else
              delpoint=1;
            end;
          end;
          if Richt==1 Input=I1(i,j+1); end;
          if Richt==3 Input=I1(i,j-1); end;
          [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
          counter,koor(counter,:)]=...
          MultiPointCal(1,Input,SW,AW,Richt,i,j,Ln,grenze,sp,ze,...
          counter,fertig,koor(anfang+1,:),...
          Konturx(Ln,Kn),Kontury(Ln,Kn));
          if delpoint==1  koordel(counter-anfang,1:2)=xy;
          else
              koordel(counter-anfang,1:2)=[0 0];
          end;
          koorex1(counter-anfang,1:2)=xy;
      end;
%-------------------------------------------------------------------
      if Richtung==[1 0 0 1]
       [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
       Kontury(Ln-dif:Ln,Kn),counter,...
       koor(counter,:)]=MultiPointCal(1,[I1(i,j+1) I1(i+1,j)],SW,AW,...
       [1 4],i,j,Ln,grenze,sp,ze,counter,...
       fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
      end;
%-------------------------------------------------------------------
      if Richtung==[1 1 1 0]
       [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
       Kontury(Ln-dif:Ln,Kn),counter,...
       koor(counter,:)]=MultiPointCal(1,[I1(i,j-1) I1(i-1,j) I1(i,j+1)],...
       SW,AW,[3 2 1],i,j,Ln,grenze,sp,ze,...
       counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
      end;
%-------------------------------------------------------------------
      if Richtung==[1 1 0 1]
       [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
       Kontury(Ln-dif:Ln,Kn),counter,...
       koor(counter,:)]=MultiPointCal(1,[I1(i-1,j) I1(i,j+1) I1(i+1,j)],...
       SW,AW,[2 1 4],i,j,Ln,grenze,sp,ze,...
       counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
      end;
%-------------------------------------------------------------------
      if Richtung==[1 0 1 1]
       [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
       Kontury(Ln-dif:Ln,Kn),counter,...
```

```
        koor(counter,:)]=MultiPointCal(1,[I1(i,j+1) I1(i+1,j) I1(i,j-1)],...
        SW,AW,[1 4 3],i,j,Ln,grenze,sp,ze,...
        counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
       end;
%-------------------------------------------------------------------------
       if Richtung==[1 1 1 1]
        [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
        Kontury(Ln-dif:Ln,Kn),counter,...
        koor(counter,:)]=MultiPointCal(1,...
        [I1(i,j+1) I1(i-1,j) I1(i,j-1) I1(i+1,j)],SW,AW,[1 2 3 4],i,j,...
        Ln,grenze,sp,ze,counter,fertig);
       end;
%-------------------------------------------------------------------------
       if Richtung==[0 1 1 0]
        [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
        Kontury(Ln-dif:Ln,Kn),counter,...
        koor(counter,:)]=MultiPointCal(1,[I1(i,j-1) I1(i-1,j)],SW,AW,...
        [3 2],i,j,Ln,grenze,sp,ze,counter,...
        fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
       end;
%-------------------------------------------------------------------------
       if Richtung==[0 1 0 1]
        %Special case
        xy=[j,i];
        hit=0;
        Richt=0;
        hit1=0;
        delpoint=0;
        [zelr splr]=size(koorex);
        for h=1:zelr
            if xy==koorex(h,1:2)
                hit=1;
                hit1=h;
            end;
        end;
        if Kontury(Ln,Kn)>i Richt=4; end;
        if Kontury(Ln,Kn)<=i Richt=2; end;
        switch hit
        case 0
         counter1=counter1+1;
         koorex(counter1,:)=[j i Richt Kn];
        case 1
         if koorex(hit1,4)==Kn
          if Richt==koorex(hit1,3)
          else
```

```
           delpoint=1;
          end;
         else
          delpoint=1;
         end;
        end;
        if Richt==2 Input=I1(i-1,j); end;
        if Richt==4 Input=I1(i+1,j); end;
        [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln,Kn),Kontury(Ln,Kn),...
        counter,koor(counter,:)]=MultiPointCal(1,Input,SW,AW,Richt,...
        i,j,Ln,grenze,sp,ze,counter,fertig,koor(anfang+1,:),...
        Konturx(Ln,Kn),Kontury(Ln,Kn));
        if delpoint==1 koordel(counter-anfang,1:2)=xy;
        else
          koordel(counter-anfang,1:2)=[0 0];
        end;
        koorex1(counter-anfang,1:2)=xy;
       end;
%-------------------------------------------------------------------------
       if Richtung==[0 1 1 1]
        [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
        Kontury(Ln-dif:Ln,Kn),counter,...
        koor(counter,:)]=MultiPointCal(1,[I1(i+1,j) I1(i,j-1) I1(i-1,j)],...
        SW,AW,[4 3 2],i,j,Ln,grenze,sp,ze,...
        counter,fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
       end;
%-------------------------------------------------------------------------
       if Richtung==[0 0 1 1]
        [dif,i,j,rv,Ln,grenze,fertig,Konturx(Ln-dif:Ln,Kn),...
        Kontury(Ln-dif:Ln,Kn),counter,...
        koor(counter,:)]=MultiPointCal(1,[I1(i+1,j) I1(i,j-1)],SW,AW,...
        [4 3],i,j,Ln,grenze,sp,ze,counter,...
        fertig,koor(anfang+1,:),Konturx(Ln,Kn),Kontury(Ln,Kn));
       end;
%-------------------------------------------------------------------------
       if fertig==0
          Richtung(1:4)=0;
          if j<sp & I1(i,j+1)<=Schwelle Richtung(1)=1; end;
          if i>1 & I1(i-1,j)<=Schwelle Richtung(2)=1; end;
          if j>1 & I1(i,j-1)<=Schwelle Richtung(3)=1; end;
          if i<ze & I1(i+1,j)<=Schwelle Richtung(4)=1; end;
          if rv==[1 0 0 0] & Richtung(1)==0 & sum(Richtung)>0 ...
                  Richtung=[0 0 0 0]; end;
          if rv==[0 0 0 1] & Richtung(4)==0 & sum(Richtung)>0 ...
                  Richtung=[0 0 0 0]; end;
```

```
          if rv==[0 0 1 0] & Richtung(3)==0 & sum(Richtung)>0 ...
                  Richtung=[0 0 0 0]; end;
          if rv==[0 1 0 0] & Richtung(2)==0 & sum(Richtung)>0 ...
                  Richtung=[0 0 0 0]; end;
       end;
      end;
     end;
     %If the special case was invoked in the contour,
     %it has to be proved if it is necessary to delete some coordinates
     %of the koor-vector, because they have to be proved once again.
%-------------------------------------------------------------------------

     if Richt>0
     [zeex spex]=size(koordel);
      for delete=(anfang+1):(zeex+anfang)
         if (koor(delete,:)==koorex1(delete-anfang,:)) & ...
                (koordel(delete-anfang,:)==[0 0])
            koor(delete,:)=[0 0];
         end;
      end;
     end;
     %Save the information of the number of contour points of the actual
     %contour.
     LN(Kn)=Ln;
    end;%ifRichtung
   end; %ifsetdif
  end;
 end;
end;
if Kn==0
    %If there was no contour found, an empty contour information should
    %be generated. This empty structure array supplies the information
    %about the impossibility to find a contour with the actual threshold.
    [mycontour(1)]=GenContour(Schwelle);
else
    for i=1:Kn
        mycontour(i)=GenContour(Schwelle,LN(i),Konturx(1:LN(i),i),...
            Kontury(1:LN(i),i));
    end;
    %If more than 2 input parameters are entered, the contours are
    %plotted.
    if nargin>2
        figure(fig);
        imagesc(I1);
        colormap(gray);
```

```
        for i=1:Kn
            hold on;
            plot(Konturx(1:LN(i),i),Kontury(1:LN(i),i),Linespec);
        end;
        hold on;
    end;
end;
%
%-------------------------------------------------------------------------
% End M file
```

```
function [Ln,Konturx,Kontury]=...
         CalContour(BerRichtung,RW,SW,AW,Ln,i,j)
%
% Use:          [Ln,Konturx,Kontury]=...
%               CalContour(BerRichtung,RW,SW,AW,Ln,i,j);
%
% Description: This routine interpolates the position of a
%               contour point with the level SW between the actual
%               value AW and the value in the direction of interest
%               RW, dependend on the direction, which is defined
%               in BerRichtung.
%
% Input Parameters:
%   BerRichtung: The information about the direction of interest.
%                Options:
%                1 : Increasing column number
%                2 : Decreasing row number
%                3 : Decreasing column number
%                4 : Increasing row number
%   RW:          The interpolation value in the direction of interest.
%   SW:          The threshold value.
%   AW:          The value of the actual position.
%   Ln:          The counter of contour points in the actual contour.
%   i:           The actual row number.
%   j:           The actual column number.
%
% Return Parameters:
%   Ln:          The counter of contour points in the actual contour.
%   Konturx:     The calculated x-position of the contour point.
%   Kontury:     The calculated y-position of the contour point.
%
% By:  Georg Egger
% Date: 10.Dec.2001
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:          Comment:
% 10.12.2001     Original Version 1.0
%--------------------------------------------------------------------

Ln=Ln+1;
```

```
switch BerRichtung
 %Interpolation of the contour point dependent on the direction.
 case 1
    Konturx=j+(AW-SW)/(AW-RW);
    Kontury=i;
 case 2
    Konturx=j;
    Kontury=i+(SW-RW)/(AW-RW)-1;
 case 3
    Konturx=j+(SW-AW)/(AW-RW);
    Kontury=i;
 case 4
    Konturx=j;
    Kontury=i+1+(RW-SW)/(AW-RW);
end;
%
%-------------------------------------------------------------------------
% End M file
```

```
function [dif,i,j,rv,Ln,grenze,fertig,Konturx,Kontury,counter,koor]=...
         MultiPointCal(lrdrehend,I1,SW,AW,Reihenfolge,i,j,Ln,grenze,sp,ze...
         ,counter,fertig,kooranfang,Konturxletzt,Konturyletzt)
%
% Use:  [dif,i,j,rv,Ln,grenze,fertig,Konturx,Kontury,counter,koor]=...
%        MultiPointCal(lrdrehend,I1,SW,AW,Reihenfolge,i,j,Ln,grenze,...
%        sp,ze,counter,fertig,kooranfang,Konturxletzt,Konturyletzt);
%
% Description: This routine calculates around the actual coordinate
%              (i,j) the contour.
%
% Input Parameters:
%   lrdrehend:   The information about the direction of rotation in the
%                actual calculation.
%   I1:          The interpolation value in the direction of interest.
%   SW:          The threshold value.
%   AW:          The value of the actual position.
%   Reihenfolge: In which direction(s) in the matrix the contour has
%                to be calculated.
%                Options:
%                1 : Increasing column number
%                2 : Decreasing row number
%                3 : Decreasing column number
%                4 : Increasing row number
%                Example: [2 3 4]
%   i:           The actual row number.
%   j:           The actual column number.
%   Ln:          The counter of contour points in the actual contour.
%   grenze:      The border of the image matrix has already been touched in
%                the direction which is written in the parameter grenze.
%   sp:          The number of columns of the matrix.
%   ze:          The number of rows of the matrix.
%   counter:     The counter of the matrix wherein the information
%                about already examined coordinates is saved(This must
%                not be coordinates of the contour!!!)
%   fertig:      indicates that the calculation of the contour in
%                one direction is finished.
%   kooranfang:  The row and column number of the startpoint of the
%                search for contour points.(optional)
%   Konturxletzt:The last x coordinate of the contour. (optional)
%   Konturyletzt:The last y coordinate of the contour. (optional)
%
% Return Parameters:
%   dif:         The number of contour points which have been calculated
%                in this function.
```

```
%   i:          The actual row number.
%   j:          The actual column number.
%   rv:         The information about the actual found direction(s)
%               of contour points lying around the actual coordinates(i,j).
%   Ln:         The counter of contour points in the actual contour.
%   grenze:     The border of the image matrix has already been touched in
%               the direction which is written in the parameter grenze.
%   fertig:     indicates that the calculation of the contour in
%               one direction is finished.
%   Konturx:    The calculated x-position or x-positions of the contour
%               point/points.
%   Kontury:    The calculated y-position or y-positions of the contour
%               point/points.
%   counter:    The counter of the matrix wherein the information
%               about already examined coordinates is saved(This must
%               not be coordinates of the contour!!!)
%   koor:       The row and column number of the last inspected matrix
%               coordinate.
%
% By:  Georg Egger
% Date: 10.Dec.2001
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:         Comment:
% 10.12.2001    Original Version 1.0
%-------------------------------------------------------------------------
points=size(Reihenfolge,2);
dif=0;
switch points
%-------------------------------------------------------------------------
case 1
  counter=counter+1;
  koor(1)=j;
  koor(2)=i;
  %Calculation of the contour point in the direction which is
  %written in Reihenfolge(1).
  [Ln,Konturx,Kontury]=CalContour(Reihenfolge(1),I1(1),SW,AW,Ln,i,j);
  %Examine if the end of the contour is already reached (for closed
  %contours only).
  if Ln>1 & kooranfang==[j i] fertig=1; end;
```

```
  %If the contour point is calculated twice, the ultimate contour
  %point is deleted.
  if Ln>1 & Konturxletzt==Konturx & Konturyletzt==Kontury Ln=Ln-1; end;
%-------------------------------------------------------------------------

case 2
  counter=counter+1;
  koor(1)=j;
  koor(2)=i;
  %Calculation of the contour point in the direction which is written
  %in Reihenfolge(1).
  [Ln,Konturx(1),Kontury(1)]=...
  CalContour(Reihenfolge(1),I1(1),SW,AW,Ln,i,j);
  %Examine if the end of the contour is already reached (for closed
  %contours only).
  if Ln>1 & kooranfang==[j i] fertig=1; end;
  %If the contour point is calculated twice, the ultimate contour point
  %is deleted.
  if Ln>1 & Konturxletzt==Konturx(1) & Konturyletzt==Kontury(1)
   Ln=Ln-1;
  end;
  if fertig==0
   %Calculation of the contour point in the direction which is written
   %in Reihenfolge(2).
   [Ln,Konturx(2),Kontury(2)]=...
   CalContour(Reihenfolge(2),I1(2),SW,AW,Ln,i,j);
   dif=1;
  end;
  Konturx=Konturx';
  Kontury=Kontury';
%-------------------------------------------------------------------------
case 3
  counter=counter+1;
  koor(1)=j;
  koor(2)=i;
  %Calculation of the contour point in the direction which is written
  %in Reihenfolge(1).
  [Ln,Konturx(1),Kontury(1)]=...
  CalContour(Reihenfolge(1),I1(1),SW,AW,Ln,i,j);
  %Examine if the end of the contour is already reached (for closed
  %contours only).
  if Ln>1 & kooranfang==[j i] fertig=1; end;
  %If the contour point is calculated twice, the ultimate contour point
  %is deleted.
  if Ln>1 & Konturxletzt==Konturx(1) & Konturyletzt==Kontury(1)
```

```
   Ln=Ln-1;
  end;
  if fertig==0
   %Calculation of the contour point in the direction which is written
   %in Reihenfolge(2).
   [Ln,Konturx(2),Kontury(2)]=...
   CalContour(Reihenfolge(2),I1(2),SW,AW,Ln,i,j);
   %Calculation of the contour point in the direction which is written
   %in Reihenfolge(3).
   [Ln,Konturx(3),Kontury(3)]=...
   CalContour(Reihenfolge(3),I1(3),SW,AW,Ln,i,j);
   dif=2;
  end;
  Konturx=Konturx';
  Kontury=Kontury';
%-------------------------------------------------------------------------
case 4
  counter=counter+1;
  koor(1)=j;
  koor(2)=i;
  %Calculation of the contour point in the direction which is written
  %in Reihenfolge(1).
  [Ln,Konturx(1),Kontury(1)]=...
  CalContour(Reihenfolge(1),I1(1),SW,AW,Ln,i,j);
  %Calculation of the contour point in the direction which is written
  %in Reihenfolge(2).
  [Ln,Konturx(2),Kontury(2)]=...
  CalContour(Reihenfolge(2),I1(2),SW,AW,Ln,i,j);
  %Calculation of the contour point in the direction which is written
  %in Reihenfolge(3).
  [Ln,Konturx(3),Kontury(3)]=...
  CalContour(Reihenfolge(3),I1(3),SW,AW,Ln,i,j);
  %Calculation of the contour point in the direction which is written
  %in Reihenfolge(4).
  [Ln,Konturx(4),Kontury(4)]=...
  CalContour(Reihenfolge(4),I1(4),SW,AW,Ln,i,j);
  %Calculation of the contour point in the direction which is written
  %in Reihenfolge(1).
  [Ln,Konturx(5),Kontury(5)]=...
  CalContour(Reihenfolge(1),I1(1),SW,AW,Ln,i,j);
  fertig=1;
  Konturx=Konturx';
  Kontury=Kontury';
  dif=4;
%-------------------------------------------------------------------------
```

```
end;
if fertig==0
  switch lrdrehend
  %Dependent on the direction of rotation the following coordinates of
  %interest are calculated.
    case 0
        switch Reihenfolge(points)
        case 1
            rv=[1 0 0 0];
            [i,j,grenze,fertig]=search(2,i,j,grenze,sp,ze,fertig);
        case 2
            rv=[0 1 0 0];
            [i,j,grenze,fertig]=search(3,i,j,grenze,sp,ze,fertig);
        case 3
            rv=[0 0 1 0];
            [i,j,grenze,fertig]=search(4,i,j,grenze,sp,ze,fertig);
        case 4
            rv=[0 0 0 1];
            [i,j,grenze,fertig]=search(1,i,j,grenze,sp,ze,fertig);
        end;
    case 1
        switch Reihenfolge(points)
        case 1
            rv=[1 0 0 0];
            [i,j,grenze,fertig]=search(4,i,j,grenze,sp,ze,fertig);
        case 2
            rv=[0 1 0 0];
            [i,j,grenze,fertig]=search(1,i,j,grenze,sp,ze,fertig);
        case 3
            rv=[0 0 1 0];
            [i,j,grenze,fertig]=search(2,i,j,grenze,sp,ze,fertig);
        case 4
            rv=[0 0 0 1];
            [i,j,grenze,fertig]=search(3,i,j,grenze,sp,ze,fertig);
        end;
    end;
else
    rv=[0 0 0 0];
end;
%
%------------------------------------------------------------------------
% End M file
```

```
function [i,j,grenze,fertig]=...
         search(SuchRichtung,i,j,grenze,sp,ze,fertig)

%
% Use:  [i,j,grenze,fertig]=...
%       search(SuchRichtung,i,j,grenze,sp,ze,fertig);
% or    [i,j,grenze]=search(SuchRichtung,i,j,grenze,sp,ze)
%
% Description: This routine calculates the new row and column number
%              of the following point of interest with respect to the
%              matrix borders.
%
% Input Parameters:
%   BerRichtung: Route of scaning.
%                1 : Increasing column number
%                2 : Decreasing row number
%                3 : Decreasing column number
%                4 : Increasing row number
%   i:           The actual row number.
%   j:           The actual column number.
%   grenze:      The border of the image matrix has already been touched in
%                the direction which is written in the parameter grenze.
%   sp:          The number of columns of the matrix.
%   ze:          The number of rows of the matrix.
%   fertig:      indicates that the calculation of the contour in
%                one direction is finished. (optional)
%
% Return Parameters:
%   i:           The actual row number.
%   j:           The actual column number.
%   grenze:      The border of the image matrix has already been touched in
%                the direction which is written in the parameter grenze.
%   fertig:      indicates that the calculation of the contour in
%                one direction is finished. (optional)
%
% By:  Georg Egger
% Date: 10.Dec.2001
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:          Comment:
```

```
% 10.12.2001     Original Version 1.0
%-----------------------------------------------------------------------

switch SuchRichtung
 case 1
    if j<sp
     %Calculating new column number.
     j=j+1;
    else
     %Saving the direction of the border in the parameter grenze.
     grenze=grenze+[1 0 0 0];
     if nargin==7 fertig=1; end;
    end;
 case 2
    if i>1
     %Calculating new row number.
     i=i-1;
    else
     %Saving the direction of the border in the parameter grenze.
     grenze=grenze+[0 1 0 0];
     if nargin==7 fertig=1; end;
    end;
 case 3
    if j>1
    %Calculating new column number.
    j=j-1;
    else
     %Saving the direction of the border in the parameter grenze.
     grenze=grenze+[0 0 1 0];
     if nargin==7 fertig=1; end;
    end;
 case 4
    if i<ze
     %Calculating new row number.
     i=i+1;
    else
     %Saving the direction of the border in the parameter grenze.
     grenze=grenze+[0 0 0 1];
     if nargin==7 fertig=1; end;
    end;
end;
%
%-----------------------------------------------------------------------
% End M file
```

```
function [startx,starty,grenze,rv1]=Startpoint(Startpunkty,...
         Startpunktx,i,j,rv1,Richtung,sp,ze,grenze,startx,starty)

%
% Use:  [startx,starty,grenze,rv1]=Startpoint(Startpunkty,...
%       Startpunktx,i,j,rv1,Richtung,sp,ze,grenze,startx,starty);
%
% Description: This routine calculates the row and column number of the
%              startpoint for the calculation of the contour in the other
%              direction in the case that the contour will not be closed.
%
% Input Parameters:
%   Startpunkty: The row number of the startpoint.
%   Startpunktx: The column number of the startpoint.
%   i:           The actual row number.
%   j:           The actual column number.
%   rv1:         The information about the actual found direction(s) of
%                contour points lying around the actual coordinates. This
%                information is necessary for the calculation of the contour
%                in the counter direction.
%   Richtung:    Route of scaning.
%                1 : Increasing column number
%                2 : Decreasing row number
%                3 : Decreasing column number
%                4 : Increasing row number
%   i:           The actual row number.
%   sp:          The number of columns of the matrix.
%   ze:          The number of rows of the matrix.
%   grenze:      The border of the image matrix has already been touched in
%                the direction which is written in the parameter grenze.
%   startx:      The column number of the startpoint for the search in
%                the counter direction.
%   starty:      The row number of the starpoint for the search in the
%                counter direction.
%
% Return Parameters:
%   startx:      The column number of the startpoint for the search in
%                the counter direction.
%   starty:      The row number of the starpoint for the search in the
%                counter direction.
%   grenze:      The border of the image matrix has already been touched in
%                the direction which is written in the parameter grenze.
%   rv1:         The information about the actual found direction(s) of
%                contour points lying around the actual coordinates. This
%                information is necessary for the calculation of the contour
```

```
%                in the counter direction.
%
% By:  Georg Egger
% Date: 10.Dec.2001
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:          Comment:
% 10.12.2001     Original Version 1.0
%-------------------------------------------------------------------------

if i==Startpunkty & j==Startpunktx
 switch Richtung
 case 1
     %Save the actual direction into rv1
     rv1=[0 1 0 0];
 case 2
     rv1=[0 0 1 0];
 case 3
     rv1=[0 0 0 1];
 case 4
     rv1=[1 0 0 0];
 end;
 %Save the column and row number of the startpoint, with respect to
 %the direction.
 [starty,startx,grenze]=search(Richtung,i,j,grenze,sp,ze);
end;
%
%-------------------------------------------------------------------------
% End M file
```

```
function mycontour=GenContour(treshold,numofpoints,xdata,ydata)
%
% Use:  mycontour=GenContour(treshold,numofpoints,xdata,ydata);
%  or   mycontour=GenContour(treshold);
%
% Description: This routine calculates a structure of important
%              informations.
%
% Input Parameters:
%   treshold:    The threshold information of the contour.
%   numofpoints: The number of points of the contour. (optional)
%   xdata:       The x coordinate of every number of the contour.
%                (optional)
%   ydata:       The x coordinate of every number of the contour.
%                (optional)
%
% Return Parameters:
%   mycontour:   The structure array.
%
% By:  Georg Egger
% Date: 10.Dec.2001
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:          Comment:
% 10.12.2001     Original Version 1.0
%-------------------------------------------------------------------------

if nargin==1
 %Calculation of an empty structure array.
 mycontour.NumberOfPoints=0;
 mycontour.xdata=[];
 mycontour.ydata=[];
 mycontour.treshold=treshold;
 mycontour.closed=[];
else
 %Calculation of the structure array.
 mycontour.NumberOfPoints=numofpoints;
 mycontour.xdata=xdata;
 mycontour.ydata=ydata;
 mycontour.treshold=treshold;
```

```
 if (xdata(numofpoints)==xdata(1)) & (ydata(numofpoints)==ydata(1))
     closed=1;
 else
     closed=0;
 end;
 mycontour.closed=closed;
end;
%
%-------------------------------------------------------------------------
% End M file
```

3.2 Konturidentifizierung

```
function [contour,fault]=CalArea(mycontour,fig,LineSpec)
%
% Use:  [contour,fault]=CalArea(mycontour);
%    or [contour,fault]=CalArea(mycontour,fig,LineSpec);
% Description: This routine identifies out of a set of contours,
%              the contour with the biggest area.
%
% Input Parameters:
%   mycontour:   The structure array.
%    +mycontour.NumberOfpoints: Number of points in the contour.
%    +mycontour.xdata:          The x-coordinates.
%    +mycontour.ydata:          The y-coordinates.
%    +mycontour.treshold:       The threshold value.
%    +mycontour.closed:         Is the contour closed? 0=No 1=Yes.
%   fig:         Handle to the figure on which the plot is to be made.
%                (optional)
%   Line Spec:   The line specification for the plot. (optional)
%
% Return Parameters:
%   contour:     The structure array.
%    +contour.NumberOfpoints: Number of points in the contour.
%    +contour.xdata:          The x-coordinates.
%    +contour.ydata:          The y-coordinates.
%    +contour.treshold:       The threshold value.
%    +contour.closed:         Is the contour closed? 0=No 1=Yes.
%   fault:       Information of possible faults:
%                0... No faults appear.
%                1... The input array was an empty array.
%                2... The input array of contours only exists of non
%                     closed contours.
%
% By:  Georg Egger
% Date: 01.02.2002
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:          Comment:
% 01.02.2002     Original Version 1.0
%-------------------------------------------------------------------------
```

```
if mycontour(1).NumberOfPoints==0
  contour=mycontour;
  fault=1;
else
 Summe1=0;
 [j i]=size(mycontour);
 for count=1:i
  if mycontour(count).closed==1
   %Implementation of the Area Calculation.
   Summe=0;
   for count1=2:mycontour(count).NumberOfPoints
    Summe=Summe+mycontour(count).xdata(count1)*...
    (mycontour(count).ydata(count1)-mycontour(count).ydata(count1-1))...
    -mycontour(count).ydata(count1)*...
    (mycontour(count).xdata(count1)-mycontour(count).xdata(count1-1));
   end;
   Summe=abs(Summe/2);
   if Summe>Summe1
       ConNumber=count;
       Summe1=Summe;
   end;
  end;
 end;
 if Summe1>0
   contour=mycontour(ConNumber);
   fault=0;
 else
   contour=mycontour;
   fault=2;
 end;
end;
if nargin>1
    figure(fig);
    hold on;
    plot(contour.xdata,contour.ydata,LineSpec);
end;
%
%-------------------------------------------------------------------------
% End M file
```

3.3 Eckfindung

```
function [Eckpunkt]=FindCorner(contour,limitcorner,fitLength,fig...
         ,Markercolor)
%-------------------------------------------------------------------------
%
% Use:  [Eckpunkt]=FindCorner(contour,limitcorner,fitLength)
%    or [Eckpunkt]=FindCorner(contour,limitcorner,fitLength,fig...
%       ,Markercolor)
% Description: This routine calculates corners of a contour.
%
% Input Parameters:
%   contour:     The structure array.
%    +contour.NumberOfpoints: Number of points in the contour.
%    +contour.xdata:          The x-coordinates.
%    +contour.ydata:          The y-coordinates.
%    +contour.treshold:       The treshold value.
%    +contour.closed:         Is the contour closed? 0=No 1=Yes.
%   limitcorner: The threshold value for the circle parameter C1.
%   fitlength:   How big has to be the datapool in which the circle
%                has to be fitted.
%   fig:         Handle to the figure on which the plot is to be made.
%                (optional)
%   Markercolor: The line specification for the plot. (optional)
%
% Return Parameters:
%   Eckpunkt:    The line coordinates of corners in the contour.
%
% By:  Georg Egger
% Date: 01.02.2002
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:          Comment:
% 01.02.2002     Original Version 1.0
%-------------------------------------------------------------------------

z2=contour.NumberOfPoints;
X1=contour.xdata;
Y1=contour.ydata;
limit=limitcorner/fitLength;
```

```
u=1;
for k=(1+z2-round(fitLength/2)):z2
    % The mean values for x and y
    % should be previously removed
    xt=X1(k:z2);
    xt(end+1:fitLength)=X1(1:k-z2+fitLength-1);
    yt=Y1(k:z2);
    yt(end+1:fitLength)=Y1(1:k-z2+fitLength-1);
    xt=xt-mean(xt);
    yt=yt-mean(yt);
    A=0;
    for i=1:fitLength
        A(i,1)=xt(i)^2+yt(i)^2;
        A(i,2)=xt(i);
        A(i,3)=yt(i);
        A(i,4)=1;
    end;
    %
    % Apply SVD to Determination of C1:C2:C3:C4
    %
    [U,S,V]=svd( A , 0 );
    C(1,k)=V(1,end);
    C(2,k)=V(2,end);
    C(3,k)=V(3,end);
    C(4,k)=V(4,end);
    %
    % Mark the segmentation points, C1 the curvature factor is used to
    % determine this point.
    %
    if abs( C(1,k) ) > limit
        Eckpunkt(u)=k+round(fitLength/2)-z2;
        u=u+1;
    end;
end;
for k=1:(z2-fitLength+1)
    % The mean values for x and y
    % should be previously removed
    xt=X1(k:k+fitLength-1);
    yt=Y1(k:k+fitLength-1);
    xt=xt-mean(xt);
    yt=yt-mean(yt);
    A=0;
    for i=1:fitLength
        A(i,1)=xt(i)^2+yt(i)^2;
        A(i,2)=xt(i);
```

```
        A(i,3)=yt(i);
        A(i,4)=1;
    end;
    %
    % Apply SVD to determine C1:C2:C3:C4
    %
    [U,S,V]=svd( A , 0 );
    C(1,k)=V(1,end);
    C(2,k)=V(2,end);
    C(3,k)=V(3,end);
    C(4,k)=V(4,end);
    %
    % Mark the segmentation points, C1 the curvature factor is used to
    % determine this point.
    %
    if abs( C(1,k) ) > limit
        Eckpunkt(u)=k+round(fitLength/2)-1;
        u=u+1;
    end;
end;
for k=(z2-fitLength+2):(z2-round(fitLength/2))
    % The mean values for x and y
    % should be previously removed
    xt=X1(k:z2);
    xt(end+1:fitLength)=X1(1:k-z2+fitLength-1);
    yt=Y1(k:z2);
    yt(end+1:fitLength)=Y1(1:k-z2+fitLength-1);
    xt=xt-mean(xt);
    yt=yt-mean(yt);
    A=0;
    for i=1:fitLength
        A(i,1)=xt(i)^2+yt(i)^2;
        A(i,2)=xt(i);
        A(i,3)=yt(i);
        A(i,4)=1;
    end;
    %
    % Apply SVD to determine C1:C2:C3:C4
    %
    [U,S,V]=svd( A , 0 );
    C(1,k)=V(1,end);
    C(2,k)=V(2,end);
    C(3,k)=V(3,end);
    C(4,k)=V(4,end);
    %
```

```
    % Mark the segmentation points, C1 the curvature factor is used to
    % determine this point.
    %
    if abs( C(1,k) ) > limit
        Eckpunkt(u)=k+round(fitLength/2);
        u=u+1;
    end;
end;
if nargin>3
 figure(fig);
 hold on;
 for i=1:length(Eckpunkt)
    plot(contour.xdata(Eckpunkt(i)),contour.ydata(Eckpunkt(i)),...
                '-.r+','LineWidth',2,...
                'MarkerEdgeColor',Markercolor,...
                'MarkerFaceColor',Markercolor,...
                'MarkerSize',15);
 end;
end;
%-------------------------------------------------------------------------
% End M file
```

3.4 Vektoren zwischen den Eckpunkten ermitteln

```
function [VektX,VektY,PunktZahl,Vektor]=VectorBetweenCorner...
         (contour,Eckpunkt,minLaenge,fig,LineSpec);
%------------------------------------------------------------------------
%
% Use:  [VektX,VektY,PunktZahl,Vektor]=VectorBetweenCorner...
%       (contour,Eckpunkt,minLaenge);
%    or [VektX,VektY,PunktZahl,Vektor]=VectorBetweenCorner...
%       (contour,Eckpunkt,minLaenge,fig,LineSpec);
%
% Description: This routine calculates vectors between corners.
%
% Input Parameters:
%   contour:     The structure array.
%    +contour.NumberOfpoints: Number of points in the contour.
%    +contour.xdata:          The x-coordinates.
%    +contour.ydata:          The y-coordinates.
%    +contour.treshold:       The treshold value.
%    +contour.closed:         Is the contour closed? 0=No 1=Yes.
%   Eckpunkt:    The line coordinates of corners in the contour.
%   minLaenge:   The minimum length of a vector.
%   fig:         Handle to the figure on which the plot is to be made.
%                (optional)
%   Line Spec:   The line specification for the plot. (optional)
%
% Return Parameters:
%   VektX:       The x-coordinates of each vector.
%   VektY:       The y-coordinates of each vector.
%   PunktZahl:   The number of points of each vector.
%   Vektor:      The line number of the startpoint and the endpoint
%                of each vector in contour.
%
% By:  Georg Egger
% Date: 01.02.2002
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:          Comment:
% 01.02.2002     Original Version 1.0
%------------------------------------------------------------------------
```

```
z2=contour.NumberOfPoints;
X1=contour.xdata;
Y1=contour.ydata;
u=1;
lEck=length(Eckpunkt);
% Determination of the line coordinate of the startpoint and the endpoint
% of each vector in contour.
for i=2:lEck
    if (Eckpunkt(i)-Eckpunkt(i-1))>minLaenge
        anfang=Eckpunkt(i-1);
        ende=Eckpunkt(i);
        Vektor(u,1)=anfang;
        Vektor(u,2)=ende;
        u=u+1;
    end;
    if i==lEck
        if Eckpunkt(i)==z2
        else
            anfang=Eckpunkt(i);
            ende=Eckpunkt(1);
            Vektor(u,1)=anfang;
            Vektor(u,2)=ende;
        end;
    end;
end;
%Determination of the vectors between the corners.
anzV=length(Vektor);
for i=1:anzV
    if i<anzV
     VX=X1(Vektor(i,1):Vektor(i,2));
     VektX(1:length(VX),i)=VX;
     VY=Y1(Vektor(i,1):Vektor(i,2));
     VektY(1:length(VX),i)=VY;
    else
     if Vektor(i,2)>Vektor(i,1)
      VX=X1(Vektor(i,1):Vektor(i,2));
      VektX(1:length(VX),i)=VX;
      VY=Y1(Vektor(i,1):Vektor(i,2));
      VektY(1:length(VX),i)=VY;
     else
      VX1=X1(Vektor(i,1):z2);
      VX2=X1(2:Vektor(i,2));
      VektX(1:length(VX1),i)=VX1;
      ENDE=length(VX1)+1;
      ENDE2=ENDE+length(VX2)-1;
```

```
      u=1;
      for g=ENDE:ENDE2
          VektX(g,i)=VX2(u);
          u=u+1;
      end;
      VY1=Y1(Vektor(i,1):z2);
      VY2=Y1(2:Vektor(i,2));
      VektY(1:length(VY1),i)=VY1;
      ENDE=length(VY1)+1;
      ENDE2=ENDE+length(VY2)-1;
      u=1;
      for g=ENDE:ENDE2
          VektY(g,i)=VY2(u);
          u=u+1;
      end;
     end;
    end;
end;
% Determination of the number of points of each vector.
for i=1:anzV
    if i<anzV
        PunktZahl(i)=Vektor(i,2)-Vektor(i,1)+1;
    else
        if Vektor(i,2)>Vektor(i,1)
          PunktZahl(i)=Vektor(i,2)-Vektor(i,1)+1;
        else
            PunktZahl(i)=Vektor(i,2)+z2-Vektor(i,1);
       end;
    end;
end;
% Plot the results.
if nargin>3
    figure(fig);
    hold on;
    for i=1:anzV
        plot(VektX(1:PunktZahl(i),i),VektY(1:PunktZahl(i),i),LineSpec);
    end;
end;
%-------------------------------------------------------------------------
% End M file
```

3.5 Eckidentifizierung

```
function [VektX1,VektY1,PunktZahl1,Vektorneu]=FindCorrectCorner...
         (contour,VektX,VektY,PunktZahl,Vektor,limitlaenge,limitdual...
         ,limitN,fig1,LineSpec1,LineSpec2);
%--------------------------------------------------------------------------
%
% Use:  [VektX1,VektY1,PunktZahl1,Vektorneu]=FindCorrectCorner...
%       (contour,VektX,VektY,PunktZahl,Vektor,limitlaenge,limitdual...
%       ,limitN);
%    or [VektX1,VektY1,PunktZahl1,Vektorneu]=FindCorrectCorner...
%       (contour,VektX,VektY,PunktZahl,Vektor,limitlaenge,limitdual...
%       ,limitN,fig1,LineSpec1,LineSpec2);
%
% Description: This routine eliminates corners of a contour, which are
%              no real corners.
%              ATTENTION. This routine can only be applied to objects,
%              which can be approximately described with lines.
%
% Input Parameters:
%   contour:     The structure array.
%    +contour.NumberOfpoints: Number of points in the contour.
%    +contour.xdata:          The x-coordinates.
%    +contour.ydata:          The y-coordinates.
%    +contour.treshold:       The treshold value.
%    +contour.closed:         Is the contour closed? 0=No 1=Yes.
%   VektX:       The x-coordinates of each vector.
%   VektY:       The y-coordinates of each vector.
%   PunktZahl:   The number of points of each vector.
%   Vektor:      The line number of the startpoint and the endpoint
%                of each vector in contour.
%   limitLaenge: The minimum length of a vector.
%   limitdual:   The threshold value for the deviation of parameter X
%                and Y between the lines.
%   limitN:      The treshold value for the deviation of parameter N
%                between the lines.
%   fig1:        Handle to the figure on which the plot is to be made.
%                (optional)
%   Line Spec1:  The line specification for the plot of the fitted lines.
%                (optional)
%   Line Spec2:  The line specification for the plot of VektX1 & VektY1
%                (optional)
%
% Return Parameters:
%   VektX1:      The x-coordinates of each new vector.
```

```
%   VektY1:      The y-coordinates of each new vector.
%   PunktZahl1:  The number of points of each new vector.
%   Vektorneu:   The line number of the startpoint and the endpoint
%                of each new vector in contour.
%
% By:  Georg Egger
% Date: 01.02.2002
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:          Comment:
% 01.02.2002     Original Version 1.0
%------------------------------------------------------------------------
z2=contour.NumberOfPoints;
X1=contour.xdata;
Y1=contour.ydata;
[ze anzV]=size(VektX);
%Fit lines to each vectorgroup.
for i=1:anzV
 one=ones(PunktZahl(i),1);
 % The mean values for x and y
 % should be previously removed
 xm=sum(VektX(1:PunktZahl(i),i))/PunktZahl(i);
 ym=sum(VektY(1:PunktZahl(i),i))/PunktZahl(i);
 VekX=VektX(1:PunktZahl(i),i)-xm;
 VekY=VektY(1:PunktZahl(i),i)-ym;
 A=[VekX,VekY,one];
 %
 % Apply SVD to determine C1:C2:C3
 %
 [U,S,V]=svd( A , 0 );
 aus=0;
 t=0;
 while aus==0
     if abs(V(3,end-t))==1
     else
         w=t;
         aus=1;
     end;
     t=t+1;
 end;
```

```
 C(1)=V(1,end-w);
 C(2)=V(2,end-w);
 C(3)=V(3,end-w);
 l2D(1)=-C(2);
 l2D(2)=C(1);
 l2D(3)=C(3)+ym*l2D(1)-xm*l2D(2);
 %Determination of the parametres X, Y, N, l to detect
 %the right datapool
 LSGYXq(1,i)=-C(2);
 LSGYXq(2,i)=C(1);
 LSGYXq(3,i)=l2D(3);
 LSGYXq(4,i)=sqrt(l2D(1)^2+l2D(2)^2);
 %Plot the fitted lines.
 if nargin>8
    range(1)=min(VektX(1:PunktZahl(i),i));
    range(2)=max(VektX(1:PunktZahl(i),i));
    range(3)=min(VektY(1:PunktZahl(i),i));
    range(4)=max(VektY(1:PunktZahl(i),i));
    plotPLine2D(fig1,l2D,LineSpec1,range);
 end;
end;
%Determination of the length of the vectors
maxlength=0;
for i=1:anzV
    LSGYXq(5,i)=0;
    for j=2:PunktZahl(i)
        LSGYXq(5,i)=LSGYXq(5,i)+sqrt((VektX(j,i)-VektX(j-1,i))^2+...
        (VektY(j,i)-VektY(j-1,i))^2);
    end;
    if round(LSGYXq(5,i))>maxlength
        maxlength=round(LSGYXq(5,i));
    end;
end;
%Determination of the right line coordinates of the vectorgroups
%to eliminate the wrong corners which have
%separated the vectors til now
[zeile spalte]=size(LSGYXq);
a=0;
for i=1:spalte
    if LSGYXq(5,i)>limitlaenge
        a=a+1;
        if a==1
            schleife=i;
        end;
    end;
```

```
end;
LSGYXq(1:5,(end+1):(end+schleife))=LSGYXq(1:5,1:schleife);
anzVneu=0;
anfang=Vektor(1,1);
ende=Vektor(1,2);
a=0;
key=0;
for i=1:(anzV+schleife)
   if LSGYXq(5,i)<limitlaenge
   else
      a=a+1;
    if a==1
      anfang=Vektor(i,1);
      ende=Vektor(i,2);
      vergl=i;
      vergl1=i;
    else
      %Comparison of X,Y,N of two lines.
      if abs(LSGYXq(1,i)-LSGYXq(1,vergl))>LSGYXq(4,vergl)/limitdual...
    & ((LSGYXq(4,i)-LSGYXq(4,vergl))/(2*limitdual)>abs(LSGYXq(1,i))...
    & abs(LSGYXq(1,i))>LSGYXq(4,vergl)/(2*limitdual))
         key=1;
      end;
      if abs(LSGYXq(2,i)-LSGYXq(2,vergl))>LSGYXq(4,vergl)/limitdual...
    & ((LSGYXq(4,i)-LSGYXq(4,vergl))/(2*limitdual)>abs(LSGYXq(2,i))...
    & abs(LSGYXq(2,i))>LSGYXq(4,vergl)/(2*limitdual))
         key=1;
      end;
      limitNo=abs(LSGYXq(3,vergl))+abs(LSGYXq(3,vergl)/limitN);
      limitNu=abs(LSGYXq(3,vergl))-abs(LSGYXq(3,vergl)/limitN);
      if abs(LSGYXq(3,i))>limitNo | abs(LSGYXq(3,i))<limitNu
         key=1;
      end;
      vergl1=vergl;
      vergl=i;
    end;
   if i==(anzV+schleife) & key==0
     anzVneu=anzVneu+1;
     Vektorneu(1,1)=anfang;
   end;
   if key==1
     ende=Vektor(vergl1,2);
     key=0;
     anzVneu=anzVneu+1;
     Vektorneu(anzVneu,1)=anfang;
```

```
      Vektorneu(anzVneu,2)=ende;
      if i==(anzV+schleife)
      else
        anfang=Vektor(i,1);
      end;
     end;
    end;
end;
%Determination of the vectors with the line coordinates from above
anzVneu=length(Vektorneu);
for i=1:anzVneu
  if Vektorneu(i,1)<Vektorneu(i,2)
     VX=X1(Vektorneu(i,1):Vektorneu(i,2));
     VektX1(1:length(VX),i)=VX;
     VY=Y1(Vektorneu(i,1):Vektorneu(i,2));
     VektY1(1:length(VX),i)=VY;
     PunktZahl1(i)=length(VX);
  else
     VX1=X1(Vektorneu(i,1):z2);
     VX2=X1(2:Vektorneu(i,2));
     PunktZahl1(i)=length(VX1)+length(VX2);
     VektX1(1:length(VX1),i)=VX1;
     ENDE=length(VX1)+1;
     ENDE2=ENDE+length(VX2)-1;
     u=1;
     for g=ENDE:ENDE2
        VektX1(g,i)=VX2(u);
        u=u+1;
     end;
     VY1=Y1(Vektorneu(i,1):z2);
     VY2=Y1(2:Vektorneu(i,2));
     VektY1(1:length(VY1),i)=VY1;
     ENDE=length(VY1)+1;
     ENDE2=ENDE+length(VY2)-1;
     u=1;
     for g=ENDE:ENDE2
        VektY1(g,i)=VY2(u);
        u=u+1;
     end;
  end;
end;
%Plot the new vectors.
if nargin==11
  figure(fig1);
  hold on;
```

```
  for i=1:anzVneu
    plot(VektX1(1:PunktZahl1(i),i),VektY1(1:PunktZahl1(i),i),LineSpec2);
  end;
end;
%
%--------------------------------------------------------------------------
% End M file
```

3.6 Linien- und Kreisanpassung

```
function [Vektor]=LineCircleDetermination(VektX1,VektY1,PunktZahl1,...
         Iterationsanzahl,fig,LineSpec);
%-------------------------------------------------------------------------
%
% Use:  [Vektor]=LineCircleDetermination(VektX1,VektY1,PunktZahl1,...
%       Iterationsanzahl);
%    or [Vektor]=LineCircleDetermination(VektX1,VektY1,PunktZahl1,...
%       Iterationsanzahl,fig,LineSpec);
%
% Description: This routine
%           1.calculates lines or circles depending on their geometry.
%           2.eliminates wrong data points with student-t distribution.
% Input Parameters:
%   VektX1:      The x-coordinates of each vector.
%   VektY1:      The y-coordinates of each vector.
%   PunktZahl:   The number of points of each vector.
%   Iterations-
%   zahl:        The number of iteration.
%   fig:         Handle to the figure on which the plot is to be made.
%                (optional)
%   LineSpec:    The line specification for the plot.
%                (optional)
%
% Return Parameters:
%   Vektor:     The structure array.
%    +Vektor.xdata:          The x-coordinates.
%    +Vektor.ydata:          The y-coordinates.
%    +Vektor.NumberOfpoints: Number of points.
%    +Vektor.LC:             Line or Circle: 1=Line 2=Circle.
%    +Vektor.X:              In case of a line it is the parameter X
%                            and in case of a circle it is the
%                            x-coordinate of the middlepoint.
%    +Vektor.Y:              In case of a line it is the parameter Y
%                            and in case of a circle it is the
%                            y-coordinate of the middlepoint.
%    +Vektor.N:              In case of a line it is the parameter N
%                            and in case of a circle it is empty.
%    +Vektor.r:              In case of a circle it is the radius of a
%                            circle and in case of a line it is empty.
%
%
% By:  Georg Egger
% Date: 01.02.2002
```

```
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:          Comment:
% 01.02.2002     Original Version 1.0
%-------------------------------------------------------------------------
for it=1:Iterationsanzahl
   [zeil,spalt]=size(VektX1);
   u=1;
   C=0;
   for k=1:spalt
    A=0;
    xt=VektX1(1:PunktZahl1(k),k);
    yt=VektY1(1:PunktZahl1(k),k);
    xm=mean(xt);
    ym=mean(yt);
    xt=xt-xm;
    yt=yt-ym;
    for i=1:PunktZahl1(k)
        A(i,1)=xt(i)^2+yt(i)^2;
        A(i,2)=xt(i);
        A(i,3)=yt(i);
        A(i,4)=1;
    end;
    %
    % Apply SVD to determine C1:C2:C3:C4
    %
    [U,S,V]=svd( A , 0 );
    C(1,k)=V(1,end);
    C(2,k)=V(2,end);
    C(3,k)=V(3,end);
    C(4,k)=V(4,end);
    %
    % Calculate the radius and middlepoint in case of a circle and
    % calculate X,Y,N in case of a line.
    %
    if C(1,k)==0
      LC(k)=1;
      X(k)=-C(3,k)
      Y(k)=C(2,k)
      N(k)=C(4,k)+ym*X(k)-xm*Y(k);
```

```
    X(k)=round(X(k)*10000)/10000;
    Y(k)=round(Y(k)*10000)/10000;
    N(k)=round(N(k)*10000)/10000;
    r(k)=0;
  else
    LC(k)=2;
    X(k)=C(2,k)/(-2*C(1,k));
    Y(k)=C(3,k)/(-2*C(1,k));
    r(k)=sqrt(X(k)^2+Y(k)^2-C(4,k)/C(1,k));
    X(k)=X(k)+xm;
    Y(k)=Y(k)+ym;
    N(k)=0;
  end;
end;
%Calculate a tool for working with statistics:
%It is the distance between the point and the line or circle
for i= 1:spalt
    if LC(i)==2;
     for j=1:PunktZahl1(i)
        Vekt0(j,i)=(VektX1(j,i)-X(i)).^2+(VektY1(j,i)-Y(i)).^2;
        Vekt0(j,i)=sqrt(Vekt0(j,i));
        Vekt1(j,i)=Vekt0(j,i)-r(i);
     end;
    else
      for j=1:PunktZahl1(i)
        if i==2
            sdfs=7;
        end;
        x=(X(i)*(VektY1(j,i)*Y(i)+VektX1(j,i)*X(i))-...
          N(i)*Y(i))/(X(i)^2+Y(i)^2);
        y=(Y(i)*(VektY1(j,i)*Y(i)+VektX1(j,i)*X(i))+...
          N(i)*X(i))/(X(i)^2+Y(i)^2);
        x1(j,i)=x;
        y1(j,i)=y;
        Vekt1(j,i)=sqrt((x-VektX1(j,i))^2+(y-VektY1(j,i))^2);
      end;
    end;
end;
studentt=[     6.314;
               2.920;
               2.353;
               2.132;
               2.015;
               1.943;
               1.895;
```

```
                    1.860;
                    1.833;
                    1.812;
                    1.796;
                    1.782;
                    1.771;
                    1.761;
                    1.753;
                    1.746;
                    1.740;
                    1.734;
                    1.729;
                    1.725];
   studentt(end+1:end+9)=1.721;
   studentt(end+1:end+10)=1.697;
   studentt(end+1:end+10)=1.684;
   studentt(end+1:end+10)=1.679;
   studentt(end+1:end+10)=1.671;
%Estimation of the sample mean value
for i=1:spalt
    xq(i)=0;
    for j=1:PunktZahl1(i)
        xq(i)=xq(i)+Vekt1(j,i);
    end;
    xq(i)=xq(i)/PunktZahl1(i);
end;
%Estimation of the sample variance and the maximal deviation.
for i=1:spalt
    Sx2(i)=0;
    for j=1:PunktZahl1(i)
        Sx2(i)=Sx2(i)+(Vekt1(j,i)-xq(i))^2;
    end;
    Sx2(i)=Sx2(i)/(PunktZahl1(i)-1);
    Sx(i)=sqrt(Sx2(i));
    if (PunktZahl1(i)-1)>60
        dev(i)=1.645*Sx(i);
    else
        dev(i)=studentt(PunktZahl1(i)-1)*Sx(i);
    end
end;
%Elimination of wrong datapoints
for i=1:spalt
    k=0;
    for j=1:PunktZahl1(i)
        if abs(Vekt1(j,i)-xq(i))>dev(i)
```

```
        else
            k=k+1;
            VektX2(k,i)=VektX1(j,i);
            VektY2(k,i)=VektY1(j,i);
        end;
    end;
    PunktZahl2(i)=k;
end;
PunktZahl1=PunktZahl2;
VektX1=VektX2;
VektY1=VektY2;
%Plot the points of the actual iteration
if nargin>4
   figure(fig);
   hold on;
   for i=1:spalt
           plot(VektX2(1:PunktZahl2(i),i),...
               VektY2(1:PunktZahl2(i),i),LineSpec);
   end;
end;

end;
%
%
   [zeil,spalt]=size(VektX1);
   u=1;
   C=0;
   for k=1:spalt
    A=0;
    xt=VektX1(1:PunktZahl1(k),k);
    yt=VektY1(1:PunktZahl1(k),k);
    xm=mean(xt);
    ym=mean(yt);
    xt=xt-xm;
    yt=yt-ym;
    for i=1:PunktZahl1(k)
        A(i,1)=xt(i)^2+yt(i)^2;
        A(i,2)=xt(i);
        A(i,3)=yt(i);
        A(i,4)=1;
    end;
    %
    % Apply SVD to determine C1:C2:C3:C4
    %
    [U,S,V]=svd( A , 0 );
```

```
    C(1,k)=V(1,end);
    C(2,k)=V(2,end);
    C(3,k)=V(3,end);
    C(4,k)=V(4,end);
    if C(1,k)==0
      LC(k)=1;
      X(k)=-C(3,k)
      Y(k)=C(2,k)
      N(k)=C(4,k)+ym*X(k)-xm*Y(k);
      X(k)=round(X(k)*10000)/10000;
      Y(k)=round(Y(k)*10000)/10000;
      N(k)=round(N(k)*10000)/10000;
      r(k)=0;
    else
      LC(k)=2;
      X(k)=C(2,k)/(-2*C(1,k));
      Y(k)=C(3,k)/(-2*C(1,k));
      r(k)=sqrt(X(k)^2+Y(k)^2-C(4,k)/C(1,k));
      X(k)=X(k)+xm;
      Y(k)=Y(k)+ym;
      N(k)=0;
    end;
   end;
 for k=1:spalt
   Vektor(k).xdata=VektX1(1:PunktZahl1(k),k);
   Vektor(k).ydata=VektY1(1:PunktZahl1(k),k);
   Vektor(k).NumberOfPoints=PunktZahl1(k);
   Vektor(k).LC=LC(k);
   Vektor(k).X=X(k);
   Vektor(k).Y=Y(k);
   if LC(k)==1
       Vektor(k).N=N(k);
       Vektor(k).r=[];
   else
       Vektor(k).N=[];
       Vektor(k).r=r(k);
   end;
end;
%--------------------------------------------------------------------------
% End M file
```

3.7 Schnittpunktermittlung

```
function [intp]=IntersPointsLineCircle(Vektor,fig,Markercolor);
%-------------------------------------------------------------------------
%
% Use:  [intp]=IntersPointsLineCircle(Vektor);
%    or [intp]=IntersPointsLineCircle(Vektor,fig,Markercolor);
%
% Description: This routine calculates intersection points between lines
%              and lines, lines and circles, circles and circles and
%              determines the correct intersection point if there are
%              more than one.
%
% Input Parameters:
%   Vektor:     The structure array.
%    +Vektor.xdata:          The x-coordinates.
%    +Vektor.ydata:          The y-coordinates.
%    +Vektor.NumberOfpoints: Number of points.
%    +Vektor.LC:             Line or Circle: 1=Line 2=Circle.
%    +Vektor.X:              In case of a line it is the parameter X
%                            and in case of a circle it is the
%                            x-coordinate of the middlepoint.
%    +Vektor.Y:              In case of a line it is the parameter Y
%                            and in case of a circle it is the
%                            y-coordinate of the middlepoint.
%    +Vektor.N:              In case of a line it is the parameter N
%                            and in case of a circle it is empty.
%    +Vektor.r:              In case of a circle it is the radius of a
%                            circle and in case of a line it is empty.
%   fig:         Handle to the figure on which the plot is to be made.
%                (optional)
%   Markercolor: The line specification for the plot.
%                (optional)
%

% Return Parameters:
%   intp:        The coordinate of the correct intersection points.
%
%
% By:  Georg Egger
% Date: 01.02.2002
% Version: 1.0
%
% (c) 2001, Institute for Automation, University of Leoben, Leoben,
% Austria
```

```
% email: automation@unileoben.ac.at, url: automation.unileoben.ac.at
%
% History:
% Date:          Comment:
% 01.02.2002     Original Version 1.0
%-------------------------------------------------------------------------
[ze spalt]=size(Vektor)
Vektor(end+1)=Vektor(1);
for i=2:(spalt+1)
  if Vektor(i-1).LC==2 & Vektor(i).LC==2
    %Determination of points of intersection circle-circle.
    if Vektor(i-1).r>Vektor(i).r
        Kreis1=Vektor(i-1);
        Kreis2=Vektor(i);
    else
        Kreis1=Vektor(i);
        Kreis2=Vektor(i-1);
    end;
    a=sqrt((Kreis1.X-Kreis2.X)^2+(Kreis1.Y-Kreis2.Y)^2);
    if (a>(Kreis1.r+Kreis2.r)) | (a<(Kreis1.r-Kreis2.r))
       intp(i-1,1:3)=0;
    elseif a==0
       intp(i-1,1:3)=0;
    else
       c=Kreis1.r;
       b=Kreis2.r;
       beta=acos((b^2-c^2-a^2)/(-2*c*a));
       gamma=acos((c^2-a^2-b^2)/(-2*a*b));
       gamma=2*pi-gamma;
       x1=0;
       y1=0;
       x2=Kreis2.X-Kreis1.X;
       y2=Kreis2.Y-Kreis1.Y;
       Y=y1-y2;
       X=x1-x2;
       N=x1*y2-x2*y1;
       %Case 1:Negativ turn
       X1beta=X*cos(-beta)+Y*sin(-beta);
       Y1beta=-X*sin(-beta)+Y*cos(-beta);
       N1beta=N+Kreis1.Y*X1beta-Kreis1.X*Y1beta;
       X1gamma=X*cos(-gamma)+Y*sin(-gamma);
       Y1gamma=-X*sin(-gamma)+Y*cos(-gamma);
       N1gamma=N+Kreis2.Y*X1gamma-Kreis2.X*Y1gamma;
       %intersection point 1:
       intp1x=...
```

```
(N1gamma*X1beta-N1beta*X1gamma)/(Y1beta*X1gamma-Y1gamma*X1beta);
intp1y=...
(N1gamma*Y1beta-N1beta*Y1gamma)/(X1gamma*Y1beta-X1beta*Y1gamma);
%Case 2: Positiv turn
X1beta=X*cos(beta)+Y*sin(beta);
Y1beta=-X*sin(beta)+Y*cos(beta);
N1beta=N+Kreis1.Y*X1beta-Kreis1.X*Y1beta;
X1gamma=X*cos(gamma)+Y*sin(gamma);
Y1gamma=-X*sin(gamma)+Y*cos(gamma);
N1gamma=N+Kreis2.Y*X1gamma-Kreis2.X*Y1gamma;

%intersection point 2:
intp2x=...
(N1gamma*X1beta-N1beta*X1gamma)/(Y1beta*X1gamma-Y1gamma*X1beta);
intp2y=...
(N1gamma*Y1beta-N1beta*Y1gamma)/(X1gamma*Y1beta-X1beta*Y1gamma);
% Calculation of the areas of two triangles given as the endpoint
% of datapool 1, the first point of datapool 2 and intersection
% point 1 or 2. The smaller area which is build with the
% correlating intersection point determines this intersection
% point as the correct one.
x1=Vektor(i-1).xdata(end);
y1=Vektor(i-1).ydata(end);
x2=intp1x;
y2=intp1y;
x3=Vektor(i).xdata(1);
y3=Vektor(i).ydata(1);
dx1=x2-x1;
dx2=x3-x2;
dx3=x1-x3;
dy1=y2-y1;
dy2=y3-y2;
dy3=y1-y3;
area1=0.5*((x1*dy1-y1*dx1)+(x2*dy2-y2*dx2)+(x3*dy3-y3*dx3));
x2=intp2x;
y2=intp2y;
dx1=x2-x1;
dx2=x3-x2;
dx3=x1-x3;
dy1=y2-y1;
dy2=y3-y2;
dy3=y1-y3;
area2=0.5*((x1*dy1-y1*dx1)+(x2*dy2-y2*dx2)+(x3*dy3-y3*dx3));
if abs(area1)>abs(area2)
    intp(i-1,1)=intp2x;
```

```
            intp(i-1,2)=intp2y;
            intp(i-1,3)=2;
        else
            intp(i-1,1)=intp1x;
            intp(i-1,2)=intp1y;
            intp(i-1,3)=2;
        end;
    end;
  end;

if (Vektor(i-1).LC==1 & Vektor(i).LC==2) | (Vektor(i-1).LC==2 & Vektor(i).LC==1)
    %Determination of points of intersection line-circle.
    if Vektor(i-1).LC==1
       Linie=Vektor(i-1);
       Kreis=Vektor(i);
    else
       Linie=Vektor(i);
       Kreis=Vektor(i-1);
    end;
    x=(Linie.X*(Kreis.Y*Linie.Y+Kreis.X*Linie.X)-Linie.N*Linie.Y)/...
      (Linie.X^2+Linie.Y^2)
    y=(Linie.Y*(Kreis.Y*Linie.Y+Kreis.X*Linie.X)+Linie.N*Linie.X)/...
      (Linie.X^2+Linie.Y^2)
    Na=sqrt((x-Kreis.X)^2+(y-Kreis.Y)^2);
    if Na>Kreis.r
       intp(i-1,1:3)=0;
    else
       if Na==Kreis.r
           intp(i-1,1)=x;
           intp(i-1,2)=y;
           intp(i-1,3)=1;
       else
           alpha=acos(Na/Kreis.r);
           %Case 1:Positiv turn -> Intersection point 1
           intp1x=x+Linie.X*tan(alpha)*(Linie.N+Kreis.X*Linie.Y-...
                  Kreis.Y*Linie.X)/(Linie.X^2+Linie.Y^2)
           intp1y=y+Linie.Y*tan(alpha)*(Linie.N+Kreis.X*Linie.Y-...
                  Kreis.Y*Linie.X)/(Linie.X^2+Linie.Y^2)
           %Case 2:Negativ turn -> Intersection point 2
           intp2x=x+Linie.X*tan(-alpha)*(Linie.N+Kreis.X*Linie.Y-...
                  Kreis.Y*Linie.X)/(Linie.X^2+Linie.Y^2)
           intp2y=y+Linie.Y*tan(-alpha)*(Linie.N+Kreis.X*Linie.Y-...
                  Kreis.Y*Linie.X)/(Linie.X^2+Linie.Y^2)
           % Calculation of the areas of two triangles given as the
           % endpoint of datapool 1, the first point of datapool 2
```

```
          % and intersection point 1 or 2. The smaller area which is
          % build with the correlating intersection point determines
          % this intersection point as the correct one.
          x1=Vektor(i-1).xdata(end);
          y1=Vektor(i-1).ydata(end);
          x2=intp1x;
          y2=intp1y;
          x3=Vektor(i).xdata(1);
          y3=Vektor(i).ydata(1);
          dx1=x2-x1;
          dx2=x3-x2;
          dx3=x1-x3;
          dy1=y2-y1;
          dy2=y3-y2;
          dy3=y1-y3;
          area1=0.5*((x1*dy1-y1*dx1)+(x2*dy2-y2*dx2)+(x3*dy3-y3*dx3));
          x2=intp2x;
          y2=intp2y;
          dx1=x2-x1;
          dx2=x3-x2;
          dx3=x1-x3;
          dy1=y2-y1;
          dy2=y3-y2;
          dy3=y1-y3;
          area2=0.5*((x1*dy1-y1*dx1)+(x2*dy2-y2*dx2)+(x3*dy3-y3*dx3));
          if abs(area1)>abs(area2)
             intp(i-1,1)=intp2x;
             intp(i-1,2)=intp2y;
             intp(i-1,3)=2;
          else
             intp(i-1,1)=intp1x;
             intp(i-1,2)=intp1y;
             intp(i-1,3)=2;
          end;
       end;
    end;
  end;
  if Vektor(i-1).LC==1 & Vektor(i).LC==1
    %Determination of points of intersection line-line.
    if Vektor(i-1).X==Vektor(i).X & Vektor(i-1).Y==Vektor(i).Y
     intp(i-1,1:3)=0;
    else
     intp(i-1,1)=(Vektor(i).N*Vektor(i-1).X-Vektor(i-1).N*...
     Vektor(i).X)/(Vektor(i-1).Y*Vektor(i).X-Vektor(i).Y*Vektor(i-1).X);
     intp(i-1,2)=(Vektor(i).N*Vektor(i-1).Y-Vektor(i-1).N*...
```

```
     Vektor(i).Y)/(Vektor(i-1).Y*Vektor(i).X-Vektor(i).Y*Vektor(i-1).X);
     intp(i-1,3)=1;
    end;
  end;
 end;
 if nargin>1
   figure(fig);
   hold on;
   [ze spalt]=size(intp);
   for i=1:ze
       plot(intp(i,1),intp(i,2),'-.g+','LineWidth',2,...
       'MarkerEdgeColor',Markercolor,...
       'MarkerFaceColor',Markercolor,...
       'MarkerSize',15);
   end;
end;
%--------------------------------------------------------------------------
% End M file
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Charles F. Van Loan Gene H. Golub. *Matrix Computations*. The Johns Hopkins University Press, Baltimore and London, ISBN 0-8018-5414-8, 1996.

[2] Hermann Grassmann. *Die Wissenschaft der extensiven Größe oder die Ausdehnungslehre, eine neue mathematische Disziplin dargestellt und durch erläutert*. Verlag von Otto Wigand, Leipzig, 1844.

[3] Felix Klein. *Elementarmathematik vom höheren Standpunkte aus*. Julios Springer Berlin, Library of Congress Catalog Card Number 67-29863, 1925.

[4] Donald E. Beasley Richard S. Figliola. *Theory and design for mechanical measurements*. John Wiley and Sons, ISBN 0-471-54441-8, 1991.

[5] H.R. Schwarz. *Numerische Mathematik*. B.G. Teubner Stuttgart, ISBN 3-519-22960-9, 1993.