Vergleichende Untersuchungen von Liniensegmentierungsverfahren insbesondere der Hough-Transformation und dem Polyline Segmentierungs-Algorithmus

Studienarbeit aus dem Gebiet der Automatisierungstechnik

vorgelegt von
Ivo Boblan
Wassermannstrasse 90
12489 Berlin
Matrikel Nr. : 132796
Fachbereich Elektrotechnik

Betreuender Assistent IPK: Dipl.-Ing. D. Sorowka
Betreuender Assistent TU: Dipl.-Ing. M. Schmauder
Betreuender Hochschullehrer TU: Prof. Dr.-Ing. R. Orglmeister

Berlin, September 1995

Inhaltsverzeichnis

1 Einleitung und Problemstellung

Im Rahmen der Entwicklung mobiler Robotersysteme spielen Liniensegmentierungsverfahren eine entscheidende Rolle. Sie sollen bestehende Navigationsverfahren stützen und verbessern. Anhand von Daten eines absolutmessenden Konturerfassungssystems (Laserscanner) werden Linien und Kanten der Umgebung in die Navigation mobiler Systeme integriert.

Die vorliegende Arbeit befaßt sich mit der vergleichenden Untersuchung von Liniensegmentierungsverfahren. Zur Beschreibung und Darstellung von Objekten und Konturen werden Liniensegmente als ein wesentliches Merkmal verwendet. Zur Ermittlung der Liniensegmente werden in der Literatur verschiedene Verfahren beschrieben, die sich in der Güte der approximierten Segmente, der Anzahl der ermittelten Segmente und im Rechenaufwand unterscheiden. Es werden die beiden Verfahren Hough-Transformation (HT) und Polyline Segmentierungs-Algorithmus (PSA) näher untersucht.

Die Arbeit soll dazu beitragen, bei der Entscheidung für das richtige Sensorsystem und bei der Wahl des geeigneten Liniensegmentierungsverfahrens zu helfen. Vergleichende Untersuchungen der beiden Verfahren werden durchgeführt und anschließend Vor- und Nachteile diskutiert. Es werden reale Meßdaten mittels eines Lasersensors genauso untersucht, wie Daten aus einem selbstentwickelten Laserdatengenerator, um genau zu prüfen, welcher Algorithmus für welche Anwendung in Frage kommt. Anhand einer entwickelten Testumgebung für beide Verfahren können unterschiedliche Parameter verändert werden, um ihren Einfluß auf die Güte der Linienerkennung und die Laufzeit deutlich zu machen.

Im Rahmen eines Projekts am Fraunhofer-Institut für Konstruktionstechnik und Produktionsanlagen in Berlin wird ein Programmpacket entwickelt und getestet, das für die Navigation eines Reinigungsroboters verwendet wird.

2 Theoretische Grundlagen

Im folgenden werden zwei unterschiedliche Verfahren zur Liniensegmentierung beschrieben. Unter Segmentierung versteht man die Unterteilung eines Bildes in Teilbereiche, von denen jeder Teilbereich bestimmte, ihn von den anderen Bereichen unterscheidende Merkmale besitzt. Die einzelnen Bildpunkte werden also durch die Segmentierung bestimmten Merkmalsklassen zugeordnet. Die Segmentierung ist ein kritischer Vorgang in einem Bilderkennungssystem, da sich Segmentierungsfehler sowohl auf die Merkmalsextraktion als auch auf die Klassifizierung auswirken können.

2.1 Hough-Transformation

Eine allgemeine Methode für das Auffinden von Konturen, über deren Lage innerhalb des betrachteten Bildes keine Information vorliegen, ist die Hough-Transformation. Sie ist eine Koordinatentransformation, die alle Punkte auf einer bestimmten Kurve (Gerade, Kreis, ...) im Bildbereich auf einen einzigen Punkt im Transformationsbereich abbildet. Ihre Unempfindlichkeit gegen Lücken innerhalb der Kontur und gegen Rauschen ist eine besonders hervorhebenswerte Eigenschaft.

2.1.1 Beschreibung des Verfahrens

Das Erkennen von Geraden ist gleichbedeutend mit dem Auffinden kollinearer Punkte. Verwenden wir zur Darstellung einer Geraden die Hesse´sche Normalform

$$x\cos(\Theta) + y\sin(\Theta) = R \qquad (2.1)$$

so ist die Gerade eindeutig durch die Parameter R und Θ bestimmt. Fällt man vom Ursprung das Lot auf die Gerade, so ist R der Abstand des Ursprungs vom Schnittpunkt des Lotes mit der Geraden und Θ der Winkel, den das Lot mit der Abszisse einschließt. Bei dieser Darstellungsform ist der Transformationsbereich (Parameterraum) die Θ-R-Ebene. Jeder Bildpunkt (x_i,y_i) wird auf der Kurve der Form (2.1) im Transformationsbereich abgebildet. Beschränken wir den Winkel auf das Intervall $0 \leq \Theta < \pi$, dann ist die Zuordnung einer Geraden im Bildbereich zu einem Punkt im Transformationsbereich eindeutig /Hesse85/.

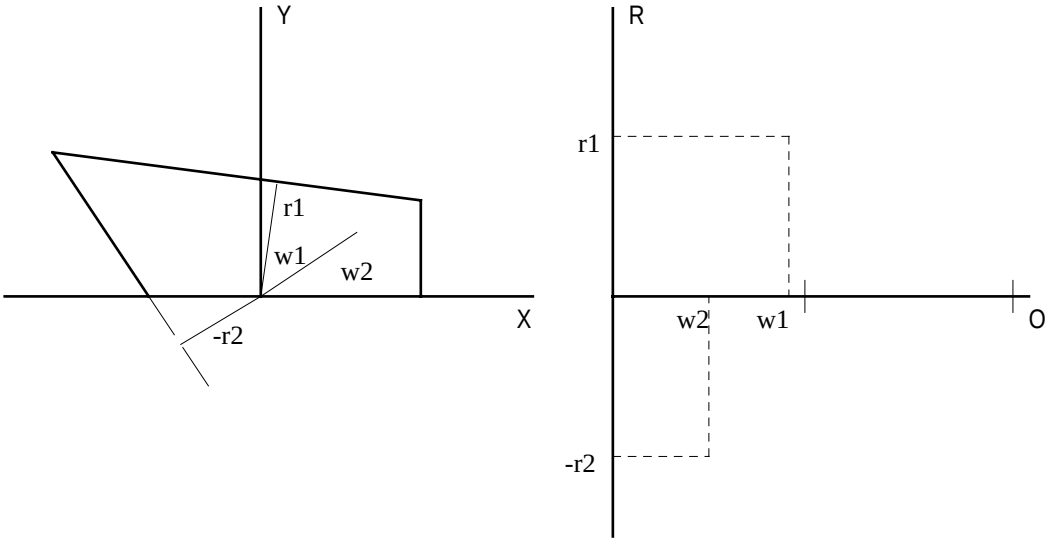

Bild 2.1 Transformation einer Geraden in den Θ-R-Raum

Jeder Punkt im Bildbereich wird zu einer sinusförmigen Kurve im Transformationsbereich. Liegen die Punkte im Bildbereich auf einer Geraden, so haben die sinusförmigen Kurven im Transformationsbereich einen gemeinsamen Schnittpunkt. Das Problem, kollineare Punkte zu finden, kann also in das Problem überführt werden, Schnittpunkte von Kurven zu finden.

Die wesentlichen Eigenschaften dieser Punkt-Kurve-Transformation können folgendermaßen zusammengestellt werden:

- ein Punkt der Bildebene entspricht einer sinusförmigen Kurve in der Transformationsebene;
- ein Punkt in der Transformationsebene entspricht einer Geraden in der Bildebene;
- den Punkten, die in der Bildebene auf einer Geraden liegen, entsprechen in der Transformationsebene Kurven, die einen gemeinsamen Schnittpunkt haben;
- Punkte in der Transformationsebene, die auf derselben Kurve liegen, sind das Ergebnis der Transformation einer durch einen Punkt laufenden Geradenschar (Geradenbüschel) in der Bildebene /Hesse85/.

Eine genaue Linienerkennung ist prinzipiell möglich, indem man nach gemeinsamen Schnittpunkten mehrerer Kurven sucht: dies erfordert allerdings einen erheblichen Rechenaufwand.

Beschränken wir uns bei der Betrachtung des Bildes auf einen kreisförmigen Ausschnitt mit dem Radius ρ, so bekommen wir im Parameterraum für R nur solche Werte, die in dem Intervall $-\rho \leq R \leq \rho$ liegen. Alle Punkte im Parameterraum, für die $R > \rho$ ist, repräsentieren Geraden außerhalb des gewählten Blickfeldes. Diese Distanzbeschränkung ist auch sehr sinnvoll, da heutige Laserscanner bei größeren Entfernungen immer ungenauer messen und eigentlich nur relativ nahe Entfernungen für einen Mobilen Roboter von Bedeutung sind. Fordern wir außerdem, daß sowohl R als auch Θ des Parameterraums nur diskrete Werte innerhalb der vorgegebenen Bereiche annehmen dürfen, so können wir auf diese Weise eine beachtliche Reduzierung des Rechenaufwandes - allerdings verbunden mit Verlust an Genauigkeit - für die Durchführung der Transformation erreichen. Mit diesen Randbedingungen können wir den betrachteten Parameterraum als zweidimensionale Anordnung von Akkumulatoren interpretieren /Hesse85/.

Ein Algorithmus zur Durchführung der Transformation kann nun wie folgt formuliert werden:

1. quantisiere den Parameterraum zwischen den Maxima und Minima der Größen Θ und R;
2. definiere ein Akkumulatorfeld $A(\Theta,R)$, dessen Elemente den Anfangswert Null haben;
3. erhöhe für jeden Bildpunkt (x,y), die Elemente des Akkumulatorfeldes um den Wert 1, die auf der zu dem Bildpunkt gehörenden Kurve liegen;
4. suche lokale Maxima innerhalb des Akkumulatorfeldes.

Die lokalen Maxima weisen auf kollineare Punkte im Bildbereich hin. Ihr Wert gibt die Anzahl dieser Bildpunkte an /Hesse85/.

Fordern wir für Θ innerhalb des Intervalls $[0,\pi]$ a äquidistante Werte und für R im Intervall $[-\rho,\rho]$ b äquidistante Werte, so besteht unser Feld aus $a \cdot b$ Elementen. Für jeden relevanten Bildpunkt (x_i,y_i) müssen nun a unterschiedliche Werte für R unter Verwendung der Gleichung (2.1) berechnet werden. Liegen insgesamt n relevante Bildpunkte vor, so muß diese Rechnung insgesamt $n \cdot a$ - mal durchgeführt werden. Anschließend müssen $a \cdot b$ Elemente auf das Vorhandensein lokaler Maxima hin untersucht werden. Der Rechenaufwand wächst also linear mit der Anzahl n der relevanten Bildpunkte.

Zusammenfassend kann festgestellt werden, daß die hier beschriebene Transformation bei der Suche nach kollinearen Punkten gute Ergebnisse liefert, wie unter Abschnitt 4.1 zu sehen ist. Liegen zusätzliche Informationen über Grenzpunkte etc. vor, so können unsinnige Ergebnisse unterdrückt werden. Die Transformationsmethode läßt sich auf Kurven, deren Form und Lage in Abhängigkeit von Parametern bestimmt sind, verallgemeinern.

2.1.2 Verallgemeinerte Formen der Hough-Transformation

Wie wir bereits im Abschnitt 2.1.1 gesehen haben, läßt sich die Hough-Transformation bei der Suche nach vorgegebenen geometrischen Formen, im dort beschriebenen Falle Geraden, verwenden. Das Ergebnis der Transformation liefert für die Parameter Θ und R, durch die die vorgegebenen Kurven beschrieben werden, eine Wertzuweisung.

Die ausführlich dargestellte Methode zum Auffinden von Geraden kann recht einfach auf allgemeinere analytisch darstellbare Kurvenformen erweitert werden. Ist die gesuchte Kurve beispielsweise ein Kreis, der in parametrischer Form durch die Gleichung

$$(x-a)^2 + (y-b)^2 = r^2 \qquad (2.2)$$

gegeben ist, so werden die Bildpunkte in den dreidimensionalen Parameterraum a-b-r transformiert. Die Parametervektoren beschreiben die Oberfläche eines Kegels /Hesse85/.

Beschränken wir die Radien der darstellbaren Kreise auf die Werte (r_k | k=1,2,...m), so besteht der Parameterraum aus m Schichten, die jeweils einen Kreis mit dem zugehörigen Radius r_k enthalten. Die Koordinaten der im Parameterraum auf dem Kreis mit dem Radius r_k liegenden Punkte sind die Koordinaten der Mittelpunkte aller möglichen Kreise mit dem Radius r_k im Bildbereich, die den untersuchten Bildpunkt enthalten. Liegen nun die untersuchten Bildpunkte auf einem Kreis mit dem Radius r_k um einen Mittelpunkt mit dem Ortsvektor (a,b), so schneiden sich im Parameterraum die Kreise mit dem Radius r_k in dem Punkt, dessen Koordinaten die Parameter des gesuchten Kreises im Bildbereich sind.

In vielen Fällen sind die Konturen der Gegenstände, die erkannt werden sollen, nicht durch Gleichungen darstellbar, so daß eine andere Referenzform für die Untersuchung gefunden werden muß. Wir können die gesuchte Kontur K als endliche Punktmenge der Form

$$K = \{\vec{x}(t) \mid t = 1,2,\ldots m\} \qquad (2.3)$$

angeben. Eine derartige Punktmenge repräsentiert z.B. einen digitalisierten Linienverlauf. Eine Verschiebung um den Vektor $\vec{r}$ ergibt die Punktmenge

$$K' = \{\vec{y}(t) | t = 1,2,\ldots m\} \tag{2.4}$$

mit

$$\vec{y}(t) = \vec{x}(t) + \vec{r}\,. \tag{2.5}$$

Das zu untersuchende Bild sei durch die Punktmenge L gegeben. In diesem Bild soll die Kontur K gefunden werden, d.h., die Kontur muß so über das Bild geschoben werden, daß eine maximale Übereinstimmung zwischen den Bildpunkten und den Konturpunkten erreicht wird /Hesse85/.

2.1.3 Fehlerbetrachtung und Korrekturverfahren

Ein Nachteil der vorgestellten Methode ist die starke Abhängigkeit der Ergebnisse von der Quantisierung des Parameterraums. Bei zu grober Quantisierung können Linien unentdeckt bleiben und die Linien die als Maxima im Akkumulatorfeld erkannt werden, haben eine zu große Abweichung von den Linien in der Bildebene. Bei zu feiner Quantisierung wächst der Rechenaufwand erheblich. Ferner kann unter Umständen das Auffinden der ´besten´ Linie dadurch verhindert werden, daß bei der Transformation die nähere Umgebung kollinearer Punkte nicht beachtet wird. Zu einer weiteren Fehlerquelle können Gruppen kollinearer Punkte werden, die für die Darstellung des Bildes ohne Bedeutung sind.

Wenn die zu untersuchenden Daten sehr verrauscht sind, treten Probleme bei der Wahl der ´richtigen´ Maxima auf. Nehmen wir an, wir scannen mit einem Laserscanner in der Bildebene zwei Linien. Die eine Linie hat 300 kollineare Punkte und die andere 50. Da unser verwendeter Laserscanner eine Meßungenauigkeit von ± 4 cm hat, gehen wir mit den verrauschten 350 Punkten in die Hough-Transformation.

Bei einer Transformation sieht ein Teil des Akkumulatorfeldes folgendermaßen aus:

R\Θ	**68**	**72**	**76**	**80**	**84**	**88**	**92**	**96**	**100**	**104**	**108**	**112**	**116**	**120**	**124**	**128**	**132**	**136**	**140**
200	7	8	8	7	8	6	2	0	0	0	0	2	4	5	5	5	5	5	4
180	8	10	10	10	9	11	9	0	0	0	0	5	5	6	7	7	7	7	7
160	11	11	13	15	16	13	11	9	0	0	6	10	11	12	12	12	12	10	10
140	11	14	20	20	23	25	31	26	9	2	23	25	25	22	18	16	13	13	12
120	17	20	22	31	32	40	50	82	131	147	87	58	43	32	28	22	18	17	13
100	19	19	22	25	34	38	39	35	20	7	32	34	32	31	26	25	24	19	18
80	15	18	18	20	20	19	18	13	3	3	7	13	16	16	17	18	16	17	16
60	15	14	14	10	11	11	9	5	2	2	2	6	8	10	12	12	13	13	14
40	10	8	8	8	6	7	6	2	3	3	3	3	5	6	7	8	9	10	10

Tabelle 2.1 Ausschnittsvergrößerung des Akkumulatorfeldes des obigen Beispiels

Die Spalten stellen die Winkelschrittweite im Akkumulatorfeld dar, welche hier 4° beträgt, die Zeilen die Distanzschrittweite mit 20 cm. Hier ist deutlich zu erkennen, daß kein klares Maximum existiert. Die Gerade mit den 50 kollinearen Punkten erkennt man hier nicht als zweites Maximum, auch wenn sie unverrauscht als Feld mit 50 Schnittpunkten existieren würde. Bei dieser doch relativ groben Auflösung existieren also schon sehr große Nebenmaxima der langen Linie, die störend auf das Erkennen von kleinen Linien wirkt. Vergrößern wir die Auflösung, so erhalten wir noch mehr störende Nebenmaxima, die allerdings nicht so groß werden.

Um eine störunanfälligere Maximaerkennung zu gewährleisten, habe ich einen Filter entwickelt, der die Nebenmaxima wegfiltert, je nach angegebener Filtergröße. Die Filtergröße sollte in Abhängigkeit der Schrittweite im Akkumulatorfeld gewählt werden. Die Filtergröße kann für den Winkel und für die Distanz getrennt vorgegeben werden. Bei zu groß gewähltem Filter kann es passieren, daß man Linien, die dicht im Akkumulatorfeld beieinander liegen, wegfiltert. Man muß also auch die Filtergröße in Abhängigkeit von der Lage der Linien in der Bildebene wählen.

Der Filter für den Winkel unterscheidet sich nur sehr wenig von dem der Distanz. Die Filtergröße für die Distanz { k=0..[1]..k_{max} } kann wahlweise von 0, keine Filterwirkung, bis

zu einer vom Anwender maximal definierten Filtergröße für die Distanz erhöht werden. Die Filtergröße für den Winkel { l=0..[1]..l_{max} } kann äquivalent zur Distanzfiltergröße gesetzt werden. Die Variablen z und s stellen die jeweiligen Zeilen- und Spaltenwerte im Akkumulatorfeld dar. Wenn z. B z_{10}=302.5 cm ist, beträgt z_9=297.5 cm und z_{11}=307.5 cm bei einer Zellgröße von 5 cm im Akkumulatorfeld. Weiterhin stellt i den Distanzschrittindex und j den Winkelschrittindex im Akkumulatorfeld dar, an deren Stelle ein Maximum gefunden wurde. Für den Zelleintrag in der μ-ten Zeile und der ν-ten Spalte steht $a_{\mu\nu}$ und $a_\nu|_{max}$ stellt den größten Zelleintrag in der ν-ten Spalte dar.

$$\bar{r} = \frac{\sum_{\nu=j-l}^{j+l}\left(\frac{\sum_{\mu=i-k}^{i+k} z_\mu a_{\mu\nu}}{\sum_{\mu=i-k}^{i+k} a_{\mu\nu}} \cdot a_\nu|_{max}\right)}{\sum_{\nu=j-l}^{j+l} a_\nu|_{max}} \tag{2.6}$$

$$\bar{\varphi} = \frac{\sum_{\mu=i-k}^{i+k}\left(\frac{\sum_{\nu=j-l}^{j+l} s_\nu a_{\nu\mu}}{\sum_{\nu=j-l}^{j+l} a_{\nu\mu}} \cdot a_\mu|_{max}\right)}{\sum_{\mu=i+k}^{i+k} a_\mu|_{max}} \tag{2.7}$$

Nach der Mittelwertbildung über μ Zeilen der ν-ten Spalte für die Distanzfilterrechnung wird mit dem Maximalwert der ν-ten Spalte gewichtet. Dann wird wieder eine Mittelwertbildung über die ν Spalten je nach Filtergröße durchgeführt. Der Filter für den Winkel ist äquivalent.

Es besteht die Möglichkeit die Filtergröße der AF-Größe anzupassen. Im Parameterfile *PARAM.C* kann man zwei Konstanten festlegen, die die Filtergröße bestimmen. Zum Einen

ist das der Filtergrößenschwellwert für die Distanz (*FILTER_SCHW_R*), und zum Anderen der Filtergrößenschwellwert für den Winkel (*FILTER_SCHW_PHI*).

Die Filter gehorchen folgenden Regeln:

Grad des Filters für die Distanz = *FILTER_SCHW_R / BBSIZE*,

Grad des Filters für den Winkel = *WSTEP / FILTER_SCHW_PHI*.

BBSIZE steht für die Größe einer AF-Zelle in Richtung der Distanz. Also bei einer Reichweite von 500 cm und einer Auflösung für die Distanz von 100 Feldern, erhalten wir für *BBSIZE* eine Größe von 5 cm. D.h. alle 5 cm beginnt im AF ein neues Feld. Wir erhalten somit, wenn wir für *FILTER_SCHW_R* einen Wert von 20 nehmen, einen ´Vier-fach-Filter´ oder besser einen 7x7-Filter für die Distanz.

Die Winkelfilterrechnung verläuft ähnlich. *WSTEP* ist, wie schon gesagt, die Winkelschrittanzahl. Nehmen wir für *FILTER_SCHW_PHI* einen Wert von 44, so bekommen wir beim letzten Durchlauf für die Winkelschrittanzahl von 176 ebenfalls einen 7x7-Filter. Für die kleineren Winkelschrittanzahlstufen ergeben sich kleinere Winkelfilter. Dies ist sinnvoll, da für kleine Werte von *WSTEP*, bei einem zu großen Filter, Linien weggefiltert werden können, die relativ dicht im AF nebeneinander liegen.

Ich habe den Filter so entworfen, daß die Felder im Akkumulatorfeld um das Hauptmaximum herum zu Null gesetzt werden. Das verfälscht natürlich Maxima, die dicht an den zu Null gesetzten Feldern stehen, da bei der Filterrechnung für Winkel und Distanz die Nullfelder miteingerechnet werden. Man könnte sich auch einen Filter vorstellen, der die Felder nicht zu Null macht, hätte dann natürlich das Problem, daß Felder mehrmals in Filterrechnungen mit eingehen. Auch würde man Nebenmaxima so nicht weg bekommen. Ein Beispiel welches die Bedeutung der Filterrechnung zeigt, könnte so aussehen:

R\Θ	**68**	**72**	**76**	**80**	**84**	**88**	**92**	**96**	**100**	**104**	**108**	**112**	**116**	**120**	**124**	**128**	**132**	**136**	**140**
200	7	8	8	7	8	6	2	0	0	0	0	2	4	5	5	5	5	5	4
180	8	10	10	10	9	11	0	0	0	0	0	0	0	6	7	7	7	7	7
160	11	11	13	15	16	13	0	0	0	0	0	0	0	12	12	12	12	10	10
140	11	14	20	20	23	25	0	0	0	0	0	0	0	22	18	16	13	13	12
120	17	20	22	31	32	40	0	0	0	147	0	0	0	32	28	22	18	17	13
100	19	19	22	25	34	38	0	0	0	0	0	0	0	31	26	25	24	19	18

80	15	18	18	20	20	19	0	0	0	0	0	0	0	16	17	18	16	17	16
60	15	14	14	10	11	11	0	0	0	0	0	0	0	10	12	12	13	13	14
40	10	8	8	8	6	7	6	2	3	3	3	3	5	6	7	8	9	10	10

Tabelle 2.2 Ausschnittsvergrößerung des Akkumulatorfeldes nach 7x7-Filterung

Man sieht deutlich, daß die großen Nebenmaxima aus Tabelle 2.1 hier zu Null gesetzt wurden. Jetzt hat auch die kleine Gerade mit 50 kollinearen Punkten eine Chance, vom Algorithmus erkannt zu werden. Wurde die Streuung zu groß gewählt, reicht ein 7x7-Filter immer noch nicht aus, um eine 50 Punkte Linie zu erkennen. Bei einer Streuung von ± 4 cm hat eine 300 Punkte Linie ein Maximum von 147, während die kleine Linie ungefähr ein Maximum von 25 besitzt. Dies reicht zum Erkennen der kleinen Linie nicht aus.

R\Θ	**12**	**16**	**20**	**24**	**28**	**32**	**36**
500	6	0	0	0	0	0	0
480	6	12	10	0	0	0	0
460	10	14	17	27	25	5	1
440	6	2	1	1	3	23	15
420	0	0	1	1	1	1	13

Tabelle 2.3 Ausschnittsvergrößerung des Akkumulatorfeldes der 50-Punkte-Linie

Man könnte natürlich kleine Linien im Akkumulatorfeld größer gewichten. Man würde dann aber auch Rauschgrößen und unsinnige Linien mehr gewichten, was zu weit verstreuten Maxima im Akkumulatorfeld führen würde.

Durch die analytische Fortsetzung der cos- und sin-Funktionen wird der softwaretechnischen Realisierung der Filterrechnung besondere Aufmerksamkeit zuteil. Gehen wir mit einer Linie in die Hough-Transformation, die einen Winkel von Null hat, also am Rand unseres Wertebereichs $0 \leq \Theta < \pi$ liegt, so haben wir auch Nebenmaxima, oder sogar das Hauptmaximum der Linie, bei $\pi - \delta$ (Delta - ein Winkelschritt) und eine negative Distanz.

2.2 Polyline Segmentierungs-Algorithmus

Bei der Polyline Segmentierung wird keine Transformation in einen Parameterraum vorgenommen. Es werden vielmehr direkt die Informationen von aufeinanderfolgenden Punkten im Bildbereich genutzt, um Linien zu erkennen.

2.2.1 Beschreibung des Verfahrens

Der Polyline Segmentierungs-Algorithmus basiert auf dem Aneinanderreihen von kollinearen Punkten, bis ein Punkt gefunden wird, der mit seinem Abstand zu den vorherigen Punkten einen Schwellwert übersteigt. Dieser Punkt wird dann zum Startpunkt für eine neue Linie. Der PSA kann besonders dann angewendet werden, wenn von einem Laserscanner aufeinanderfolgende Punkte zur Verfügung gestellt werden, um die Information der nebeneinander liegenden Punkte zu nutzen. Die beiden nachfolgenden Bilder demonstrieren anschaulich den PSA.

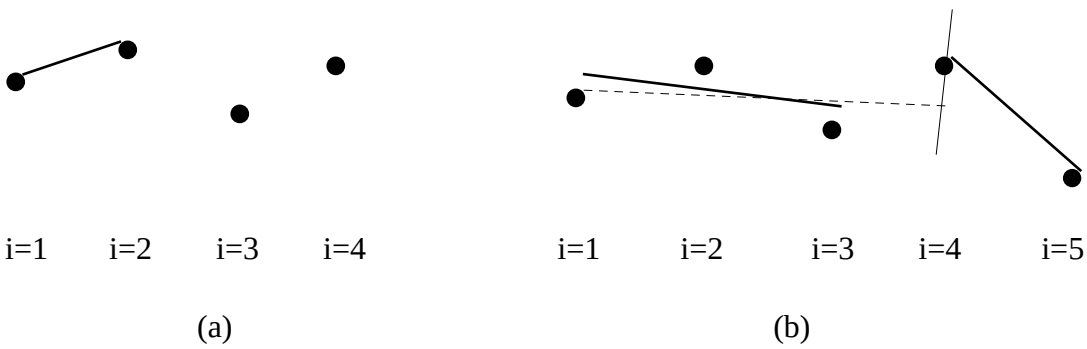

Bild 2.2 Idee des PSA (Polyline Segmentierungs-Algorithmus)

Beim PSA werden zu Beginn die beiden ersten Punkte miteinander verbunden. Die Linie wird so zum dritten Punkt verlängert, daß die Distanz von der Linie zu jedem der drei Punkte minimiert wird. Man sieht in Bild 2.2 b, daß die ´schwebenden´ Endpunkte der Linie den Einfluß des Startes in Bild 2.2 a verringern. Die gepunktete Linie in Bild 2.2 b be-

schreibt die Position der Linie, die entsteht, wenn man den vierten Punkt zur Linie addiert. Wenn aber die Distanz des vierten Punktes Δ_4 zur existierenden Linie einen Schwellwert übersteigt, wird eine neue Linie erstellt, die mit dem vierten Punkt startet.

Wir definieren eine Linie *L*, welche die Punkte *n*, *n*+1,..., *n*+*k*-1 verbindet, als:

$$L = (\rho, \Theta)_{n,k} \tag{2.8}$$

Weiterhin definieren wir eine Entstellung (Distorsion) für die Linie *L* als:

$$DIST_{n,k} = \frac{1}{k-1} \sum_{i=n}^{n+k-1} \Delta_i^{\,2} \tag{2.9}$$

Der Polyline Segmentierungs-Algorithmus kann nun wie folgt dargestellt werden, wenn man *m* Punkte miteinander verbinden will.

1. $n = 1$
2. $k = 2$
3. Erstellen der Linie zwischen Punkt *n* und *n*+1
4. Berechne die Linie $(\rho, \Theta)_{n,k+1}$ und minimiere $DIST_{n,k+1}$
5. IF ($DIST_{n,k+1}$ < SCHWELLE) AND (*n*+*k* < m)
   k = *k*+1
   $L = (\rho, \Theta)_{n,k}$
   GOTO 4

ELSE

IF ($n+k$ < m-1)

$n = n+k$

GOTO 2

ELSE

HALT

Der einzige nicht triviale Schritt, ist die Berechnung der Distanz im Punkt 4 /Ekman/.

Man kann beim PSA sehr gut die Abhängigkeit der Linienerkennung gegenüber der Rauschgröße anhand der Schwelle einstellen. Dort wo Segmente mit geschlossenen Koordinaten in andere Segmente übergehen, könnte man z.B. den Algorithmus weniger empfindlich machen gegenüber Rauschen. Die rekursive Form des PSA ermöglicht eine schnelle Ausführung, unabhängig von der Anzahl der Punkte.

Es folgt jetzt die Herleitung der Gleichungen die für das Errechnen der Koordinaten der Linie (p,Θ) benötigt werden /Duda73/.

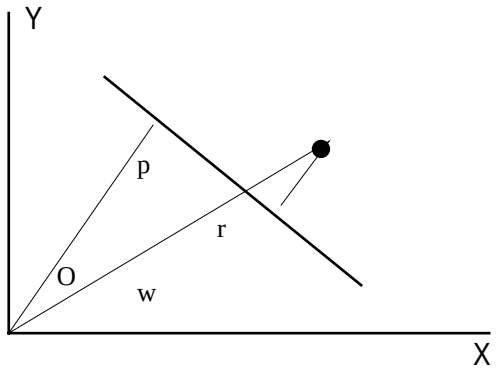

Bild 2.3 Darstellung der Distanz von einem Punkt zur entsprechenden Linie

Δ_i kann geschrieben werden als (vgl. Bild 2.3):

$$\Delta_i = r_i \left(\cos(\varphi_i) \cos(\Theta) + \sin(\varphi_i) \sin(\Theta) \right) - p \qquad (2.10)$$

In dieser Gleichung kann $\cos(\varphi_i)$ und $\sin(\varphi_i)$ im Vorfeld berechnet und in eine Tabelle der Größe 3200*2 abgelegt werden. Die Tabellengröße kann natürlich beliebig verändert werden. Im weiteren schreiben wir anstatt $\cos(\varphi_i)$ c_i und für $\sin(\varphi_i)$ s_i , wobei $\varphi_i = i \cdot 2\pi / 3200$ gesetzt wird.

Wenn wir die Gleichung (2.10) in die Gleichung (2.9) einsetzen, ist es möglich den Ausdruck der Distorsion in folgender Weise zu schreiben:

$$\begin{aligned} DIST_{n,k} = \frac{1}{k-1}\Bigg(& k \cdot p^2 + \cos^2\Theta \cdot \sum_{i=n}^{n+k-1} (r_i c_i)^2 + \sin^2\Theta \cdot \sum_{i=n}^{n+k-1} (r_i s_i)^2 + \\ & 2\sin\Theta\cos\Theta \cdot \sum_{i=n}^{n+k-1} r_i^2 c_i s_i - 2p\left(\cos\Theta \cdot \sum_{i=n}^{n+k-1} r_i c_i + \sin\Theta \cdot \sum_{i=n}^{n+k-1} r_i s_i \right) \Bigg) \end{aligned} \qquad (2.11)$$

Führen wir für die jeweiligen Summen A, B, C, D und E ein, erhalten wir:

$$\begin{aligned} DIST_{n,k} = \frac{1}{k-1}\Big(& k \cdot p^2 + \cos^2\Theta \cdot A + \sin^2\Theta \cdot B + \\ & 2\sin\Theta\cos\Theta \cdot C - 2p\left(\cos\Theta \cdot D + \sin\Theta \cdot E\right)\Big) \end{aligned} \qquad (2.12)$$

Als nächstes wird der Ausdruck so umgeformt, daß p isoliert in der Gleichung steht.

$$\begin{aligned} DIST_{n,k} = \frac{k}{k-1}\Bigg(& \left(p - \frac{\cos\Theta \cdot D + \sin\Theta \cdot E}{k} \right)^2 + \frac{1}{k}\Bigg(\cos^2\Theta \cdot \left(A - \frac{D^2}{k} \right) + \\ & \sin^2\Theta \cdot \left(B - \frac{E^2}{k} \right) + 2\sin\Theta\cos\Theta \cdot \left(C - \frac{DE}{k} \right) \Bigg) \Bigg) \end{aligned} \qquad (2.13)$$

Der quadratische Ausdruck mit *p* ist immer ≥ 0. Daraus kann man schließen, daß man hier die Distorsion minimieren kann. Das optimale *p* ($\hat{p}$) kann gewählt werden als (wenn das optimale Θ ($\hat{\Theta}$) bekannt ist):

$$\hat{p} = \frac{\cos\hat{\Theta}\cdot D + \sin\hat{\Theta}\cdot E}{k} \tag{2.14}$$

Die Gleichung (2.14) kann wieder formuliert werden als:

$$\hat{p} = \frac{1}{k}\sum_{i=n}^{n+k-1} r_i \cdot \cos\left(\hat{\Theta} - \varphi_i\right) \tag{2.15}$$

Hier sieht man deutlich, daß *p* der arithmetische Mittelwert der Orthogonalprojektion der Meßpunkte entlang einer Linie ist. Die Winkelkomponente Θ ist somit gleich $\hat{\Theta}$. Es ist leicht zu sehen, daß *p* nicht negativ werden kann.

Im Folgenden wird angenommen, daß *p* selektiert wird wie in Gleichung (2.14) und Gleichung (2.13) wird umgeschrieben in:

$$\begin{aligned} DIST_{n,k} = \frac{1}{k-1}\Bigg(& \cos^2\Theta\cdot\left(A - \frac{D^2}{k}\right) + \\ & \sin^2\Theta\cdot\left(B - \frac{E^2}{k}\right) + 2\sin\Theta\cos\Theta\cdot\left(C - \frac{DE}{k}\right)\Bigg) \end{aligned} \tag{2.16}$$

Als nächstes definieren wir H, I und J als:

$$H = A - \frac{D^2}{k}, \qquad I = B - \frac{E^2}{k}, \qquad J = C - \frac{DE}{k} \tag{2.17}$$

welche eingesetzt, zur folgenden Beschreibung der Distorsion führen:

$$DIST = \frac{1}{k-1}\left(\cos^2\Theta \cdot H + \sin^2\Theta \cdot I + 2\sin\Theta\cos\Theta \cdot J\right) \tag{2.18}$$

Das Minimum kann jetzt gefunden werden, durch die Ableitung der Distorsion nach Θ. Zuvor schreiben wir Gleichung (2.18) noch etwas um.

$$DIST = \frac{1}{k-1}\left(\frac{1+\cos(2\Theta)}{2} \cdot H + \frac{1-\cos(2\Theta)}{2} \cdot I + \sin(2\Theta) \cdot J\right) \tag{2.19}$$

(2.19) vereinfachen wir weiter zu:

$$DIST = \frac{1}{k-1}\left(\cos(2\Theta) \cdot \frac{H-I}{2} + \frac{H+I}{2} + \sin(2\Theta) \cdot J\right) \tag{2.20}$$

Die Ableitung errechnet sich dann zu:

$$\frac{\partial DIST}{\partial \Theta} = \frac{2}{k-1}\left(\cos(2\Theta) \cdot J - \sin(2\Theta) \cdot K\right) \tag{2.21}$$

wobei *K* definiert ist als:

$$K = \frac{H - I}{2} \tag{2.22}$$

Setzt man nun weiter $2\Theta = q$ und die Ableitung = 0, erhält man:

$$\cos(q) \cdot J = \sin(q) \cdot K \tag{2.23}$$

Im Intervall $[0,2\pi]$, kommen nun für q zwei mögliche Lösungen in Frage:

$$q = a\tan\left(\frac{J}{K}\right) + \nu \cdot \pi \qquad \nu \in Z \tag{2.24}$$

welche uns dann vier Lösungen für Θ gibt:

$$\Theta = \frac{1}{2} \cdot a\tan\left(\frac{J}{K}\right) + \nu \cdot \frac{\pi}{2} \qquad \nu \in Z \tag{2.25}$$

Wir definieren jetzt eine Funktion *W*:

$$W = \begin{cases} 0 & K > 0 \\ \pi & K < 0 \end{cases} \tag{2.26}$$

(Der Fall K = 0 muß speziell behandelt werden.)

Nun kann *q* eindeutig im Intervall $[0,2\pi]$ ausgedrückt werden in folgender Form:

$$q = a\tan\left(\frac{J}{K}\right) + W \tag{2.27}$$

Und Θ ist dann:

$$\Theta = \frac{1}{2}\left(a\tan\left(\frac{J}{K}\right) + W \pm \pi\right) \tag{2.28}$$

Jeden dieser zwei Werte, kann man in Gleichung (2.18) einsetzten, und es kommt der gleiche Wert für DIST heraus. Also kann man dann einen der zwei Werte aus Gleichung (2.28) nehmen für die Berechnung von DIST.

Nach der DIST-Berechnung vergleichen wir diesen Wert mit einer Schwelle, die wir uns selbst vorgeben. Ist also der orthogonale Abstand eines Punktes kleiner als unsere vorgegebene Schwelle, der Punkt gehört also noch zur Linie, so ergibt sich $\hat{p}$ mit Θ für $\hat{\Theta}$ aus der Gleichung (2.14).

Für die Berechnung von $\hat{p}$ ist die Wahl von Θ sehr wichtig. Setzt man ein falsches Θ in Gleichung (2.14) ein, erhält man ein negatives $\hat{p}$. Erhält man also ein positives $\hat{p}$, so hat man das richtige Θ gewählt, während ein negatives $\hat{p}$ zu einer Addition von 180° zu Θ führt. Die Koordinaten der Linie werden jetzt mit $(\hat{p},\hat{\Theta})$ aktualisiert (Bild 2.2b).

Ist nun andererseits DIST größer als die Schwelle, wird eine neue Linie erstellt, die mit diesem Punkt startet (Bild 2.2b). Hier sehen wir schon den großen Vorteil beim PSA gegenüber der Hough-Transformation, er erkennt Lücken im Liniensegment z.B. Türen in einer Wand.

Wie die Herleitung der Rechnung erkennen läßt, ist es möglich, den PSA rekursiv arbeitend umzuformen. Nachfolgend stehen fünf rekursive Summen, die für die Variablen verwendet werden:

$$\begin{cases} A_{n,k} = 0 & k = 0 \\ A_{n,k} = A_{n,k-1} + \left(r_n \cdot c_n\right)^2 & k = 1,2,\ldots \end{cases}$$

$$\begin{cases} B_{n,k} = 0 & k = 0 \\ B_{n,k} = B_{n,k-1} + \left(r_n \cdot s_n\right)^2 & k = 1,2,\ldots \end{cases}$$

$$\begin{cases} C_{n,k} = 0 & k = 0 \\ C_{n,k} = C_{n,k-1} + r_n^{\,2} \cdot c_n \cdot s_n & k = 1,2,\ldots \end{cases}$$

$$\begin{cases} D_{n,k} = 0 & k = 0 \\ D_{n,k} = D_{n,k-1} + r_n \cdot c_n & k = 1,2,\ldots \end{cases}$$

$$\begin{cases} E_{n,k} = 0 & k = 0 \\ E_{n,k} = E_{n,k-1} + r_n \cdot s_n & k = 1,2,\ldots \end{cases}$$

Die weiter oben gesetzten *J* und *K* können wie folgt berechnet werden:

$$J_{n,k} = C_{n,k} - \frac{D_{n,k} E_{n,k}}{k} \tag{2.29}$$

$$K_{n,k} = \frac{1}{2}\left(A_{n,k} - \frac{D_{n,k}^{\,2}}{k} - B_{n,k} + \frac{E_{n,k}^{\,2}}{k} \right) \tag{2.30}$$

Es ist möglich die Berechnung von *J* und *K* auch in rekursive Ausdrücke zu schreiben, aber der Gewinn in Gleichmäßigkeit übersteigt nicht die Degradation der Ausführungsgeschwindigkeit. /Ekman/

2.2.2 Fehlerbetrachtung

Für den Polyline Segmentierungs-Algorithmus ist entscheidend, die Schwelle richtig zu wählen. Wenn man nun eine Laserungenauigkeit von ± 4 cm hat, wird es sinnvoll sein, die Schwelle auf jeden Fall größer als die Laserstreuung zu setzen. Da DIST den quadratischen Abstand zwischen aktuellem Punkt und momentan extrahierter Linie darstellt, ist die Schwelle auch das Quadrat der Abweichung. Setzt man die Schwelle kleiner, so werden nicht existierende Linien erkannt. Aufeinanderfolgende Punkte können in diesem Fall maximal <8 cm auseinander liegen, d.h., vier oder fünf als Wert für die Streuung in diesem Beispiel ist ein guter Wert. Wird die Schwelle größer gewählt, gehen die ersten Punkte der neuen Linie noch in die alte Linie mit ein. Es ergibt sich eine Verfälschung der Hesse-Parameter (p,Θ) der neuen Linie. Es besteht auch die Möglichkeit, daß eine folgende kleine Linie dann nicht mehr erkannt wird. Man sollte also die Schwelle auch im Bezug zur Scanumgebung wählen. Weitere Erläuterungen werden im Abschnitt 4.2 gegeben.

3 Implementierung

3.1 Beschreibung der verwendeten Hardware

Es wurde als Computer ein INTEL PC 486-25 MHz verwendet und auf der Basis des Betriebssystems MS-DOS in Borland C programmiert. Man kann sich hier natürlich auch jede andere Hardwarebasis vorstellen, unter einem anderen Betriebssystem und in einer anderen Sprache. Sinnvoll wäre es bei der Hough-Transformation, da die Abarbeitung des Akkumulatorfeldes doch sehr rechenintensiv ist, ein Transputersystem oder ein anderes Multiprozessorsystem einzusetzen. Beide Systeme erlauben eine gleichzeitige Abarbeitung verschiedener Programmteile, dadurch wird wertvolle Programmlaufzeit gespart. Um die Hough-Transformation in einem echtzeitfähigen System sinnvoll einsetzen zu können, müssen hohe Anforderungen an Speicherplatz und Schleifenoptimierung gesetzt werden. Diese lassen sich bei geschickter Wahl der Hardwarekomponenten z.B. durch RISC-Prozessoren erfüllen.

3.1.1 Beschreibung der Sensorhardware

Als Meßsystem wird ein tastender Laserscanner PLS (programmierbarer Laserscanner) der Firma SICK mit einem neuartigen Schutzsystem verwendet. Es vermißt auf der Basis der Lichtlaufzeitmessung seine Umgebung wie ein Radargerät. Das Spezielle und Neuartige daran ist, daß es keine Reflektoren benötigt, sondern - wie Radar auch - tastend arbeitet. Erst die tastende Funktion erlaubt die individuelle Einstellung der Überwachungsbereiche. Das wahlweise einstellbare vorgelagerte Warnfeld sorgt für die frühe Vorwarnung bereits vor der Unterbrechung des eigentlichen Schutzfeldes. Unabhängig davon können die stetig anfallenden Meßdaten über die Umgebungskontur für Meßaufgaben ausgewertet werden. Die Speicherung des Warnfeldes kann - abhängig vom Anwendungsfall - speicherresistent (im EPROM) oder flüchtig im RAM geschehen. Beim fahrerlosen Transportsystem (FTS),

das sein Warnfeld in Größe und Form stetig an den Fahrkurs anpassen soll, erfolgt die Speicherung im RAM.

Der PLS ist für die Anwendung in geschlossenen Räumen konzipiert (d.h. es sollte kein Nebel, Regen, Schnee oder starker Rauch vorhanden sein). Er eignet sich zum Einsatz in Fertigungsanlagen und an Flurförderzeugen. Weitere Anwendungsgebiete sind:

- Innenraumabsicherung an Werkzeugmaschinen (z.B. Pressen)
- Lageerkennung von Objekten
- Größenvermessung von Objekten
- Formerkennung
- Querschnittvermessung

Im Anwendungsfeld muß der PLS so montiert werden, daß das gewünschte Schutzfeld in einem Halbkreis (max. 180°) und mit einem Radius von max. 4 m erfaßbar ist. Bei der Auswahl der günstigsten Sensorposition ist zu beachten, daß

- statische Hindernisse möglichst keine Schlagschatten verursachen sollten
- die Montageposition möglichst optimal für die elektrische Installation ausfällt
- der Sensor möglichst geschützt positioniert wird.

Der PLS benötigt zur Stromversorgung eine Gleichspannung von 24 V ±15% mit einer Leistung von 14 W zzgl. der Last an den drei möglichen Ausgängen (sie beträgt max. 2*250 mA und 1*100 mA). Das Gerät verfügt über einen Schnittstellenanschluß RS 422 oder RS 232.

Der PLS hat drei Leuchtmelder. Nur der rote und der grüne haben Bedeutung für das Schutzfeld. Der gelbe Leuchtmelder signalisiert u.a. eine Verletzung das Warnfeldes, aber auch Verschmutzung.

Der Laserscanner PLS ist ein Reflexions-Lichttaster, der nach dem Prinzip der Lichtlaufzeitmessung arbeitet. Ein sehr kurzer Lichtimpuls wird ausgesandt. Gleichzeitig wird eine „elektronische Stoppuhr“ gestartet. Über einen rotierenden Spiegel wird der Lichtstrahl abgelenkt und so eine halbkreisförmige Fläche überstrichen. Trifft das Licht auf ein Hindernis, wird das diffus reflektierte Licht vom Sensor empfangen und die „Stoppuhr“ angehalten. Aus der verstrichenen Zeit zwischen Sende- und Empfangssignal wird die Entfernung zum reflektierenden Objekt errechnet. Mit der zum Meßstrahl zugehörigen Winkelinformation kann auch die genaue Position des Objektes bestimmt werden. Liegt die gemessene Entfernung unter der für den Meßstrahl eingestellten kritischen Entfernung für des Schutzfeld bzw. für das Warnfeld, schaltet der PLS die Maschine ab oder löst ein Warnsignal aus.

Der Sicherheits-Laserscanner eignet sich zur Bereichsabsicherung an gefährlichen stationären Maschinen und Anlagen und als Kollisions- und Personenschutz an fahrerlosen Transportfahrzeugen (FTS). Beim FTS dient das Schutzfeld zum Not-Aus des Fahrzeuges, wenn ein Objekt oder eine Person innerhalb der eingestellten Grenzen detektiert wird. Wird ein Objekt innerhalb des Warnfeldes detektiert, kann das Fahrzeug kontrolliert bis in die Langsam- oder Schleichfahrt abbremsen und ein Warnsignal auslösen. Ein PLS kann - bei FTS-Einsatz - so eingestellt werden, daß er nach einem Not-Aus selbständig nach n Sekunden weiterfährt.

Zur Navigationsunterstützung kann der PLS ebenfalls eingesetzt werden, da er unabhängig vom Überwachungsmodus, stetig die Entfernung und die Winkelposition zu Objekten in seinem Sichtfeld mißt. Über die 180° eines Scans kommen so 361 Koordinatenmeßwerte zustande, die die Umgebungskontur in der Scanebene widerspiegeln. Autonome Fahrzeuge, die über ein internes Navigationssystem verfügen, können diese Daten zum Update ihres Navigationssystems nutzen. Der PLS ist über eine serielle Verbindung (RS 232 oder RS 422) fest an einen Auswertungsrechner anzuschließen.

Zu guter Letzt noch einige technische Daten:

- **Schutzfeld**

Reichweite:	4 m Radius
Ansprechzeit:	< 80 ms

Mindest-Remission: 1.8 %, diffus

Auflösung: > 70 mm

- **Warnfeld**

  Reichweite: ca. 15 m Radius

- **Meßbereich**

  Reichweite: max. 50 m

  Auflösung

  Entfernungsmessung: ±40 mm

  Winkelauflösung: 0.5 °

  Scanzeit: 40 ms

- **Allgemeine Daten**

  Scanwinkel, max.: 180°

  Laserschutzklasse: 1

  Betriebsumgebungstemperatur: 0...+50 °C

  Lagertemperatur: -25...+70 °C

  Meßfehler: typ. ±50 mm

  worst case: max. 94 mm in 2 m Abstand

  max. 131 mm in 4 m Abstand

  Sender: Infrarot-Laserdiode

  Öffnungswinkel Empfänger: ±1 °

  Gewicht: 4.5 kg

3.2 Software

Als Betriebssystem wurde MS-DOS verwendet, welches ein Single Task System ist. Programmiert wurden die Algorithmen in Borland C mit dem Borland Compiler 2.0. Es besteht kein Problem, die Software in Microsoft C zu compilieren, wenn man kleine Änderungen in der Grafikansteuerung durchführt und einige Funktionsnamen umbenennt.

3.2.1 C - Implementierung

Die Algorithmen wurden komplett in der Programmiersprache C implementiert, Durch einen strukturierten Programmaufbau wird die Lesbarkeit wesentlich erhöht. Im folgenden Headerfile sind Datenstrukturen, Typendefinitionen und Prototypenvereinbarungen zusammengestellt.

Typen:

```
typedef struct bild_ber
{
        double phi;
        double r;
} BILD_BEREICH;

typedef struct trans_ber
{
        double phi;
        double r;
        int anzahl;
} TRANS_BEREICH;

typedef int huge AKKUMULATOR_FELD[182][500];
#define IN
```

#define OUT

Funktionen:

*void GetParameter(IN char *parameterfilename);*

*void SetParameter(IN char *parameterfilename ,*

*IN char *xydatenfilename ,*

*IN/OUT char *parameterstring);*

*void ConvertXYToPhiR(IN char *xydatenfilename ,*

*OUT BILD_BEREICH *bb ,*

*OUT TRANS_BEREICH *hessebb ,*

*OUT double *hessebbphi ,*

*OUT char *phirdatenfile);*

void SetStreuung(IN double streuung ,

*IN/OUT BILD_BEREICH *bb);*

*void PutBildGraph(IN BILD_BEREICH *bb);*

*void CalcAkkumulatorFeld(IN BILD_BEREICH *bb ,*

OUT AKKUMULATOR_FELD af);

void GetMaxima(IN AKKUMULATOR_FELD af ,

*OUT TRANS_BEREICH *focus);*

*void Filter(IN char *was , // über phi, r oder beide*

IN int momentanwinkelschritt ,

IN int momentandistanzschritt ,

IN/OUT AKKUMULATOR_FELD af ,

```
                OUT TRANS_BEREICH *focus );


void PutHesseValue(  IN int xpos ,
                     IN int ypos ,
                     IN TRANS_BEREICH *focus );
void PutResultGraph( IN TRANS_BEREICH *focus );
double Guetekriterium(       IN TRANS_BEREICH *bb ,
                             IN TRANS_BEREICH *focus );
void PutGueteGraph( IN int xpos ,
                    IN int ypos ,
                    IN char *gütefilename ,
                    IN char *algorithmus ,                  // hough oder psa
                    IN int anzahldurchläufe );
void CalcPSA(IN BILD_BEREICH *bb ,
                OUT TRANS_BEREICH *focus );
```

Eine Hesse´sche Gerade ist charakterisiert durch ihre orthogonale Distanz zum Ursprung und ihren Winkel zur waagerechten Achse des Koordinatensystems. Dies wurde berücksichtigt in den Typen *BILD_BEREICH* und *TRANS_BEREICH*. Beide vereinen den Winkel und die Entfernung einerseits der Geraden im Bildbereich, also die nominalen oder vorgegebenen, und andererseits der Geraden im Transformationsbereich, also die geschätzten oder durch den Algorithmus errechneten Geraden. Der Integer-Wert *anzahl*, zählt die Schnittpunkte der sinusförmigen Kurven im Akkumulatorfeld bei der Hough-Transformation.

Das große Problem bei der Hough-Transformation, ist das relativ groß zu wählende Akkumulatorfeld (AF). Je größer das AF, desto besser die Auflösung, desto genauer die geschätzten Geraden. Unser AF mit der Größe 182*500**int*, belegt im Hauptspeicher 182*500*2 = 182 Kilobyte (K). Dadurch erreicht man eine Auflösung von 2 cm und 1°. Für eine bessere Auflösung, also für genauere Rechnungsergebnisse, muß man einen größeren Hauptspeicher zur Verfügung stellen. Um ein Datenfeld in C unter DOS größer als 64 K anzulegen, muß man es mit *typedef int huge* vereinbaren. Wenn man es dann mit *farcalloc(...)* speichermäßig anmeldet, kann man wie gewohnt auf das zweidimensionale Feld zugreifen.

Die grobe Struktur des Programms zeigt folgendes Bild:

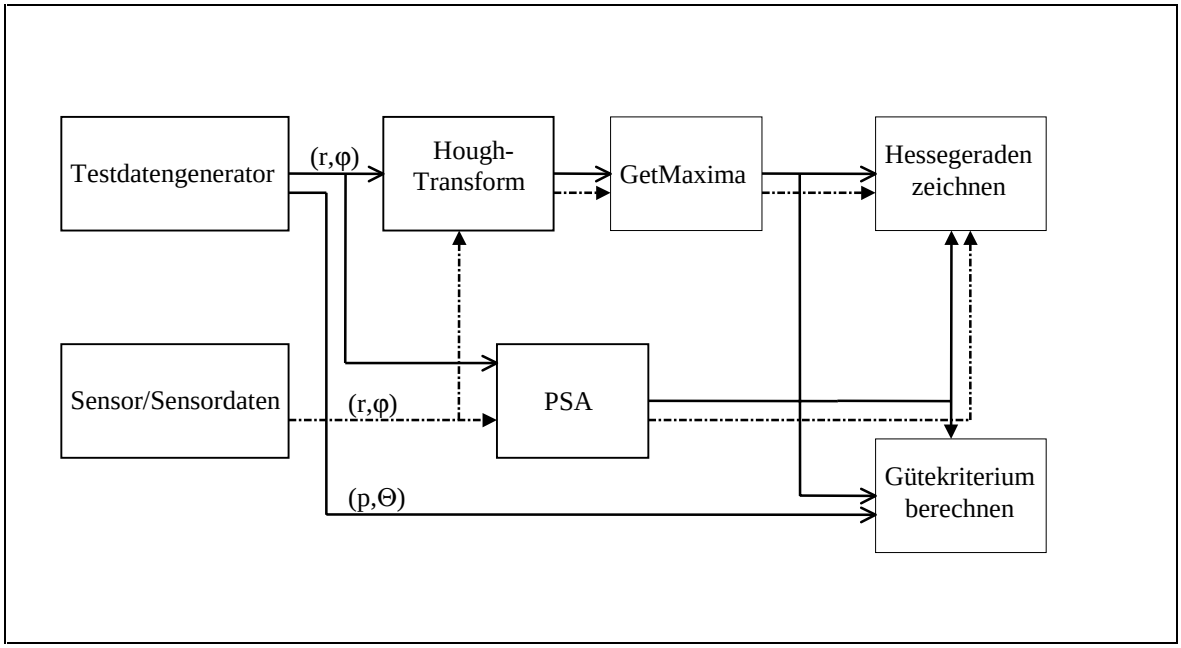

Bild 3.1 Programmstruktur der wichtigsten Programmteile

Es muß, beim Start des Programms als erster Kommandozeilenparameter der verwendete Algorithmus angegeben werden, Hough oder PSA. Groß- oder Kleinschreibung spielt hier keine Rolle. Der zweite Kommandozeilenparameter ist optional anzugeben (vgl. Abschnitt 3.2.2). Die Funktionen *GetParameter(...)* und *SetParameter(...)* dienen zum Einlesen und Schreiben eines Parameterfiles. *ConvertXYToPhiR(...)* und *SetStreuung(...)* sind Funktionen

des Testdatengenerators, welcher in Abschnitt 3.2.2 beschrieben wird. *PutBildGraph(...)* zeichnet die Liniendaten des Bildbereiches einschließlich eines Koordinatensystems.

CalcAkkumulatorfeld(...) beinhaltet den eigentlichen Algorithmus der Hough-Transformation. Hier werden zwei ineinander verschachtelte Schleifen durchlaufen, um jedes Feld im Akkumulatorfeld (AF) ansprechen zu können. Es werden die jeweiligen Felder hochgezählt, um später eine Aussage treffen zu können, wieviele Schnittpunkte pro Feld, aufgetreten sind. Die Folgefunktion *GetMaxima(...)* sucht anschließend das komplette AF nach vorhandenen Maxima ab, und schreibt die so gefundenen Hesse-Parameter in das Feld *focus[i]* .Wurde im Parameterfile eine Filtergröße für den Winkel und/oder eine Filtergröße für die Entfernung angegeben, so wird vor *GetMaxima(...)* in der Funktion *Filter(...)* eine Filterrechnung (vgl. Abschnitt 2.1.3) durchgeführt. Diese wird notwendig, da sehr große Nebenmaxima auftreten können, die dazu beitragen, daß Linien mit wenig kollinearen Punkten nicht mehr durch *GetMaxima(...)* erkannt werden. Der Filter wurde schon in Abschnitt 2.1.3 näher vorgestellt.

CalcPSA(...) beinhaltet die mathematische Formulierung des Polyline Segmentierungs-Algorithmus. In der Funktion ist das unter Abschnitt 2.2.1 vorgestellte rekursive Update-Schema implementiert. Erst werden die fünf Summen berechnet, um dann nacheinander die Hesse-Parameter zu erhalten. Wird ein Punkt gefunden, der nicht mehr zur Linie gehört, wird dieser als Startpunkt für die nächste Linie verwendet. Somit ist es möglich Lücken im Linienverlauf zu erkennen. Im Abschnitt 4.2 werden die durchgeführten Tests dazu beschrieben.

Die Funktionen *PutHesseValue(...)* und *PutResultGraph(...)* dienen zur Anzeige der gefundenen Hesse-Parameter und zur Grafikausgabe der gefundenen Linien. Durch Vorgabe der Linien mit einem Testdatengenerator, sind die nominalen Hesse-Parameter im Ausgangsbildbereich bekannt. Diese kann man nun mit den gefundenen Parametern der Linien, wie sie aus den jeweiligen Algorithmen kommen, vergleichen. Diese Aufgabe übernimmt im Programm die Funktion *Guetekriterium(...)*. Sie erzeugt ein Gütemaß, welches in eine Datei geschrieben wird und später ausgewertet werden kann. Das Gütekriterium sieht folgendermaßen aus:

$$J = \sum_{i=1}^{\text{Geradenanzahl}} \left(\left(\hat{r}_i - r_i \right)^2 + \left(\left(\hat{\varphi}_i - \varphi_i \right) \cdot r_i \right)^2 \right) \qquad (3.1)$$

Das Gütekriterium wurde von mir so entworfen, um in gleinem Maße den Distanzfehler und den Winkelfehler bewerten zu können. Der Winkel muß in Rad umgerechnet werden, damit die Beziehung (3.1) gilt. Der Winkelfehler wird noch mit der nominalen Distanz multipliziert, damit die Einheiten und Dimensionen übereinstimmen. Es müssen natürlich, besonders wichtig bei der Hough-Transformation, die gefundenen Linien in der Reihenfolge der vorgegebenen Linien sortiert werden. Je kleiner das Gütemaß, um so kleiner ist die Abweichung der gefundenen zu den vorgegebenen Linien, desto besser ist der jeweilige Algorithmus. Wird zu einer Linie im Bildbereich keine passende oder eine falsche Linie gefunden, wird das Gütekriterium immer größer und gleichzeitig zum Maß für die Qualität der Linienerkennung. Die Güte J veranschaulicht die Summe der quadratischen Abweichungen und hat die Einheit cm^2. *PutGueteGraph(...)* zeichnet eine anschauliche Grafik, welche die Abhängigkeit des Gütemaßes von den jeweiligen Parametern verdeutlicht.

3.2.2 C - Testumgebung

Mit der Funktion *SetParameter(...)* kann man aber auch die Punktwerte im Testdatengeneratorfile *DAT_XY.C* verändern. Dieses File wird vom Testdatengenerator eingelesen, für einen Testdurchlauf, bei dem sich z.B. eine Tür im Liniensegment eines Raumes öffnet. Der zweite Kommandozeilenparameter könnte wie folgt aussehen, *LINE3_AY=100[10]300* . Hier würde von der dritten Linie, vom Anfangspunkt der Y-Wert von 100 cm in Zehnerschritten bis 300 cm verändert werden. Weiterhin können die Parameter im File *PARAM.C* verändert werden in der Notation *STREUUNG=1[1]10*. Dieser Ausdruck als zweiter Kommandozeilenparameter würde veranlassen, daß das Programm zehnmal durchlaufen wird, mit einer Streuung von eins in Einerschritten bis zehn.

Durch die integrierte Testumgebung wird es möglich, einzelne Parameter zu verändern, um ihre Abhängigkeit auf das Gütemaß zu erhalten. Parameter können z.B. sein; *STREUUNG*, *SCHWELLE* für den PSA, *WSTEP* oder *BBSIZE* zur Dimensionierung des AF im

Houghtransformationsalgorithmus, u.v.m. .Am Ende jedes Durchlaufs werden das Gütemaß, die benötigte Zeit des Rechenalgorithmus und die Winkel zwischen den Geraden in seperate Files geschrieben. Diese können dann beliebig weiterverarbeitet werden (vgl. Abschnitt 4).

3.2.3 Optimierung

Die Hough-Transformation (HT) benötigt sehr viel Zeit, da jeder Bildpunkt erst in den Parameterraum transformiert werden muß, aus dem man dann nach längerer Suche die Hesse-Parameter erhält. Der PSA hingegen rechnet in weit unter 50 ms diese Geradenparameter aus.

Man kann nun natürlich, um die Berechnung schneller ablaufen zu lassen, zum Einen eine schnellere Hardware einsetzen und zum Anderen die Programmierung im Bezug auf die Laufzeit optimieren.

Eine schnellere Hardware erschöpft sich schnell in den Kriterien ökonomischer und technologischer Rentabilität. Natürlich wird durch große SPARC oder RISC-Maschinen die Programmlaufzeit weiter verringert, doch ist diese Hardware dann nur schlecht auf einen mobilen Roboter zu positionieren.

Bei der Programmierung jedoch kann man relativ leicht eine erhebliche Laufzeiteinsparung erzielen. Zeitangaben, welcher Programmschritt welchem vorzuziehen ist, sind allgemein bekannt und unterscheiden sich von Prozessor zu Prozessor nicht im wesentlichen. Unterschiede bestehen dann erst bei RISC-Prozessoren oder großen Main-Frame-Rechnern. Für den Einsatz des Compiler gilt im allgemeinen: Testcompilieren mit einem gängigen Borlandcompiler, da kurze Compilierungszeiten erzielt werden. Die Endversion sollte dann aber doch mit einem Microsoftprodukt bearbeitet werden, weil das fertige Programm schneller läuft, und der Compiler in bezug auf Programmgröße bessere Werte liefert.

Es sollte aber unbedingt hardwarenah programmiert werden. Das verschlechtert zwar die Portabilität, verringert aber die Programmlaufzeit erheblich. Nicht endende *IF ... THEN ...*

ELSE Verzweigungen sollten genauso wenig benutzt werden, wie zeitaufwendige Funktionen aus der *printf(...)*-Familie.

4 Simulationstechnische Erprobung und Gegenüberstellung der Ergebnisse

Im folgenden wird versucht, die Stärken und Schwächen der einzelnen Verfahren herauszufinden. Das geschieht am Besten, indem gemessene Daten gegenüberstellt, oder Grafiken erstellt werden, an Hand derer man dann Schlüsse über die Genauigkeit der Rechnung oder über die Anwendbarkeit des Algorithmus ziehen kann. Ich habe bei beiden verwendeten Verfahren folgende Abhängigkeiten gegenübergestellt:

- Güte(Parameter)
- Laufzeit(Parameter)

  Hough-Parameter: - WSTEP (Winkelschrittanzahl im AF)

  - STREUUNG (Punktabweichung von kollinearen Punkten für den Testdatengenerator)

  - Winkel (Winkel zwischen zwei aufeinanderfolgenden Linien für Testdatengenerator (TDG))

  - Linienlänge (Länge einer zu erkennenden Linie)

  PSA-Parameter: - SCHWELLE (Schwellwert, zum Erkennen einer neuen Line)

  - STREUUNG und Winkel wie oben

Weitere Wahloptionen liegen in der Anzahl der Scanpunkte des Testdatengenerators, in der Quantisierungsgröße für die Distanz im AF, in der Entfernung der im Bildbereich abgescannten Linien vom Ursprung und in der Filtergröße bei der Hough-Transformation.

4.1 Hough-Transformation

Zu Beginn wird bei der Hough-Transformation die Güte (Gleichung 3.1) in Abhängigkeit von der Winkelschrittanzahl im AF (*WSTEP*) untersucht:

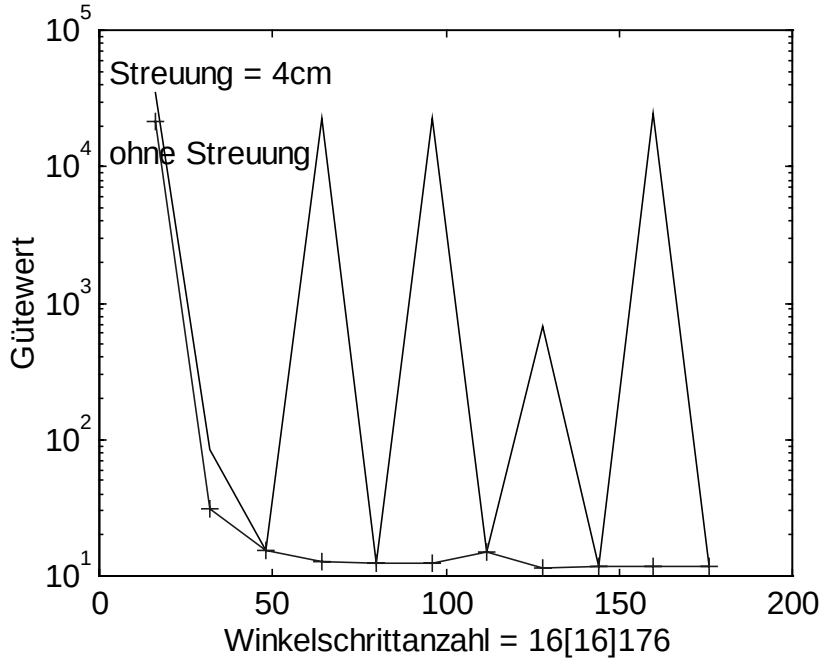

Bild 4.1 Gütewert in Abhängigkeit von der Winkelschrittanzahl im AF

Die mit Kreuzen markierte Linie zeigt die genannte Abhängigkeit bei einer Streuung von Null. Hier sieht man gut, daß nach einer gewissen AF-Vergrößerung, im Bezug auf die Winkelschrittanzahl, keine wesentliche Verbesserung der Güte zu erzielen ist. Diese Tatsache relativiert sich natürlich wieder, wenn man berücksichtigt, daß es Laserdaten ohne Streuung nicht gibt. Bei einer angenommenen Streuung von ± 4 cm, das liegt im Bereich des von uns verwendeten Lasers (vgl. Abschnitt 3.1.1), sieht die Abhängigkeit schon wesentlich ruppiger aus. Bei einem Programmdurchlauf ist eine Güte unverhältnismäßig schlecht, und beim Nächsten liegt sie wieder nahe der Güte mit der Streuung gleich Null. Dieser unruhige Verlauf liegt weniger an der zu hohen Streuung, als vielmehr am schlechten Quantisierungsverhältnis. Ein schlechtes Quantisierungsverhältnis liegt dann vor, wenn

durch die Einteilung in feste Spalten und Zeilen, das errechnete Maximum in eine angrenzenden Spalten oder Zeile neben das theoretische Maximum fällt. Diese Eigenschaft im quantisierten AF ist mit einer Filterrechnung über die angrenzenden Spalten und Zeilen zu verhindern. Man kann also bei einer Streuung ungleich Null, den Hough-Transformations-Algorithmus nicht ohne weiteres verwenden (vgl. Bild. 4.1). Es sollte immer eine Filterrechnung durchgeführt, oder mindestens die Quantisierung im AF so fein wie möglich gewählt werden (sehr hoher Speicheraufwand).

Im nächsten Bild ist die Laufzeit dargestellt, mit der selben Winkelschrittanzahl wie in Bild 4.1.

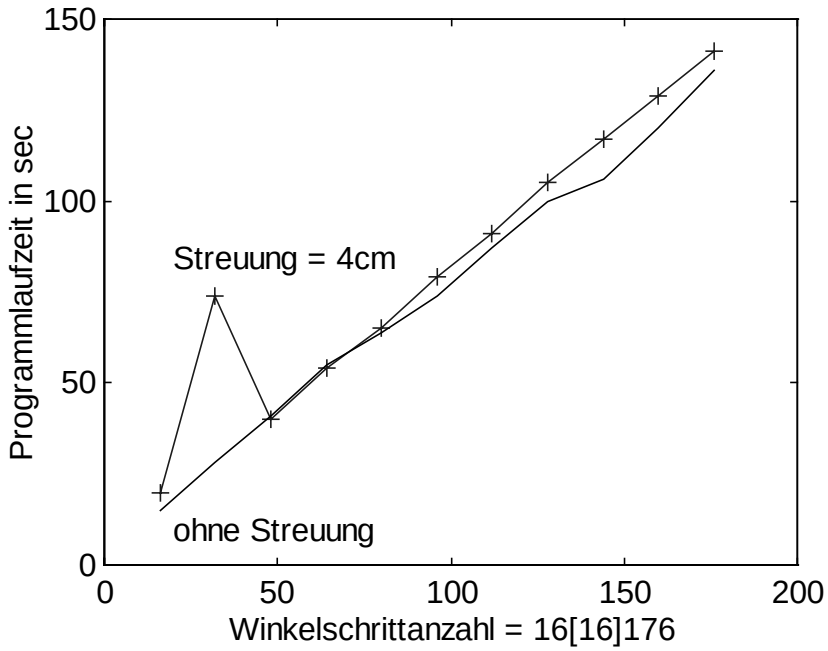

Bild 4.2 Programmlaufzeit in Abhängigkeit von der Winkelschrittanzahl im AF

Man sieht hier sehr schön, daß die Programmlaufzeit fast linear mit der Vergrößerung des AF steigt. Es existiert hier ein minimaler Unterschied zwischen dem Durchlauf mit Streuung und dem ohne Streuung. Es werden bei einer Streuung ungleich Null auch Nebenmaxima als Hauptmaxima erkannt. Dadurch wird die genaue Zuordnung zu den nominalen Hesse-Parametern erschwert. Diese Zuordnung ist notwendig, damit die Gütefunktion richtige Werte liefert. Der kurze Anstieg der Programmlaufzeit bei der Winkelschrittanzahl

von 32 erklärt sich aus der Tatsache, daß hier bei einer zu geringen Quantisierung des AF und relativ großer Streuung der Punktdaten, die Funktion zur Erkennung der Maxima nicht schnell genug die Hauptmaxima findet und zuordnet. Erforderlich ist also eine Mindestgröße des AF, um ein schnelles Finden und Zuordnen der Hesse-Parameter zu garantieren.

Im dritten Bild konzentrieren wir uns wieder auf den Gütewert. Hier wird dieselbe Abhängigkeit dargestellt, allerdings mit variablem Filter. Variabler Filter heißt, die Filtergröße paßt sich der gegebenen AF-Größe an (vgl. Abschnitt 2.1.3).

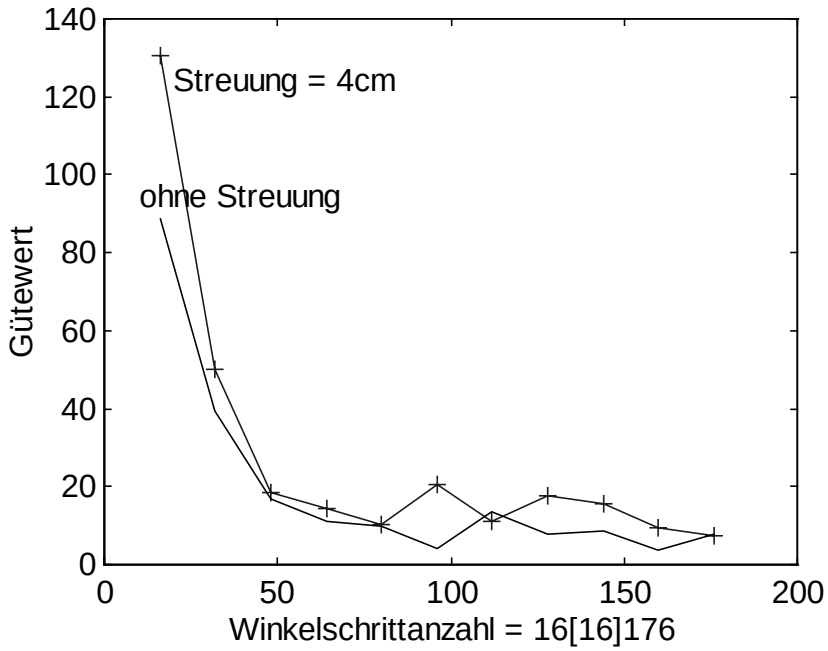

Bild 4.3 Gütewert in Abhängigkeit von der Winkelschrittanzahl mit variablem Filter

Hier sieht man deutlich, daß auch für eine große Streuung der Gütewert mit der vorgestellten Filterrechnung klein gehalten wird. Bei Verwendung eines Filters spielt also die Streuung der Laserdaten keine so entscheidende Rolle mehr. Hier ist auch wieder deutlich zu sehen, daß eine gewisse AF-Größe eingehalten werden muß, um eine gute Linienerkennung zu erhalten. Es stellt sich natürlich die Frage, wie groß die Streuung maximal werden darf, und ob die Filterrechnung auch negative Auswirkungen hat.

Betrachten wir hier wieder die Programmlaufzeit, ist leicht einzusehen, gegenüber Bild 4.2, daß der Filter die Suche nach den passenden Linien erleichtert.

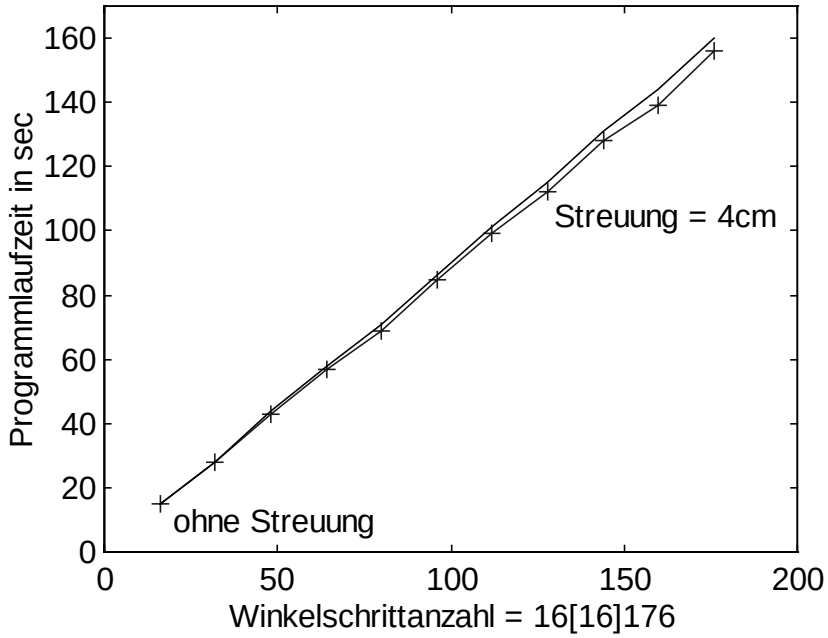

Bild 4.4 Laufzeit in Abhängigkeit von der Winkelschrittanzahl mit variablem Filter

Hier ist gut zu erkennen, daß bei einem Durchlauf mit Filter, die Streuung keinen Einfluß auf die Programmlaufzeit ausübt. Wir erhalten einen fast linearen Verlauf der Programmlaufzeit bei linearer Erhöhung des AF. Das ist auch verständlich, da jedes Feld (Zelle) im AF durchlaufen werden muß.

Vergleichen wir weiterhin die Programmlaufzeiten mit und ohne Filterrechnung bei einer Streuung von ±4 cm, erhalten wir folgende Abhängigkeit in Bild 4.5.

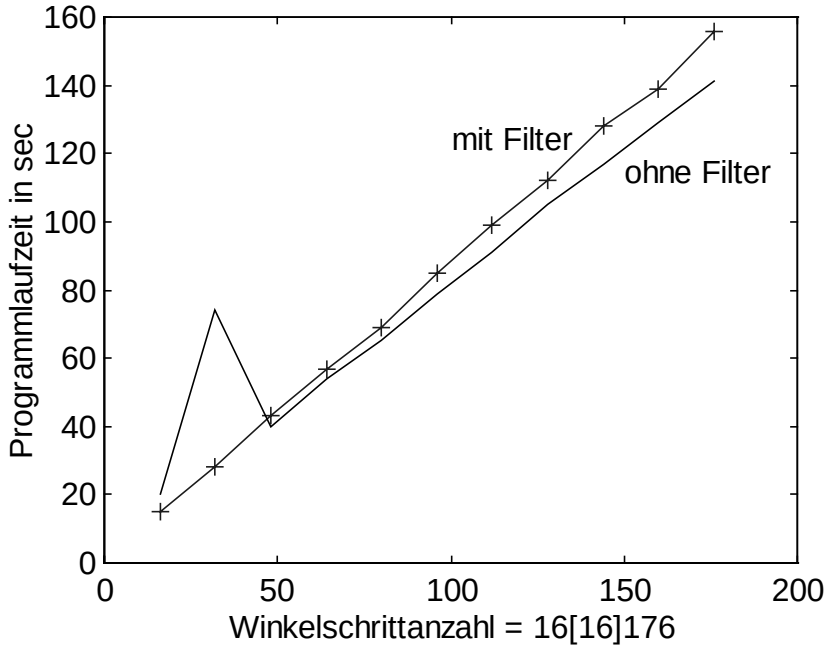

Bild 4.5 Laufzeit in Abhängigkeit der Winkelschrittanzahl mit und ohne Filter

Hier sind noch mal in einer Grafik vereint die Kurve aus Bild 4.2 ohne Filter und die Kurve aus Bild 4.4 mit Filterrechnung. Die Filterrechnung trägt also zur Laufzeiterhöhung bei. Je größer das AF wird, desto größer wird der Unterschied, da jedes Feld im AF auch im Bezug auf mögliche Filterrechnung durchsucht wird.

Desweiteren wollen wir die Beziehung zwischen der Streuung und dem Gütewert untersuchen.

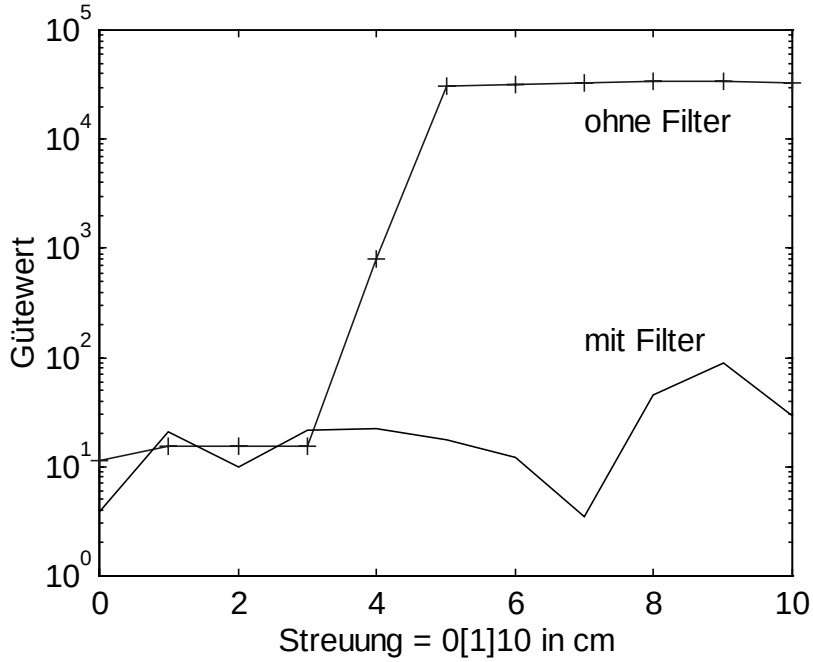

Bild 4.6 Gütewert in Abhängigkeit von der Streuung mit und ohne Filter

Hier ist gut zu sehen, daß bei einer weiteren Erhöhung der Streuung, bei einer Rechnung ohne Filter, der Gütewert maximal wird, d.h. es erfolgt keine Linienerkennung mehr. Will man die Hough-Transformation ohne Filterrechnung sinnvoll einsetzen, dürfen die Punktdaten eines Sensors nicht mehr als mit ± 3 cm verrauscht sein. Wenn man mit Filter die Transformation durchführt, bekommt man auch noch bei einer Streuung von ± 10 cm einen passablen Gütewert heraus. Die Filterrechnung ist, wie man in Bild 4.6 sieht, noch nicht die Beste, da doch ein relativ unruhiger Verlauf zu erkennen ist. Um eine sichere Linienerkennung zu gewährleisten, ist zu empfehlen, immer eine Filterrechnung durchzuführen.

Untersuchen wir nun die Abhängigkeit der Güte von dem eingeschlossenen Winkel zweier zu erkennender Linien im Bildbereich, erhalten wir folgendes Bild.

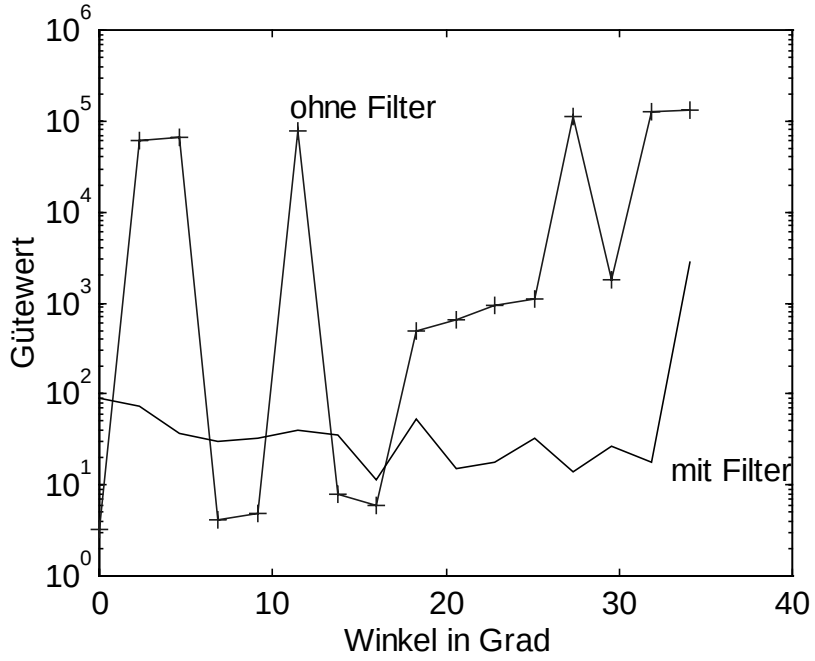

Bild 4.7 Gütewert in Abhängigkeit vom Winkel zwischen zwei Linien mit und ohne Filter

Dieser Testdurchlauf klärt für uns die Frage, ob und nach welchem Winkel der Algorithmus zwei Linien erkennt, die aus einer Linie durch konstante Winkelveränderung von 180° bis 145° entstehen. In der Grafik steht 0° für 180°, also für den Fakt, daß beide Linien die selben Hesse-Parameter besitzen. Bei einem Winkel von Null hat die Kurve ohne Filter einen kleineren Gütewert als die mit Filter. Das wird unmittelbar klar, da ohne Filterrechnung viele Nebenmaxima existieren, von denen das Größte für die zweite Linie genommen wird. Die Kurve mit Filter hat einen größeren Gütewert bei 0°, da die großen Nebenmaxima weggefiltert werden, und die dann noch existierenden kleinen Nebenmaxima schon relativ weit entfernt von den wahren Parametern liegen. Für die Kurve ohne Filter ist wieder der typisch unruhige Verlauf zu erkennen (vgl. Bild 4.1).

Der Kurvenverlauf mit Filterrechnung orientiert sich im Verlauf der Erhöhung des Winkels in Richtung kleinerer Gütewerte. Das wird unmittelbar deutlich, wenn man sich vorstellt, daß die Maxima der beiden Linien im AF dicht beieinanderliegen und mit größer werden-

dem Winkel weiter auseinander rücken. Liegt nun das zweite Maximum am Filterrand des ersten Maximums, gehen für die Filterrechnung des Zweiten, die durch die erste Filterrechnung gesetzten Nullen im AF mit ein und verfälschen die Parameter des zweiten Maximums. Erst wenn beide Maxima so weit auseinander liegen, daß sie ihre Filterrechnung unabhängig voneinander durchführen können, erhält man genaue Hesse-Parameter für beide Linien. Es ist also hier schon zusehen, daß die Filtergröße auch in Abhängigkeit von der Linienanordnung im Bildbereich gewählt werden muß.

Das nachfolgende Bild verdeutlicht die Abhängigkeit der Güte von der Anzahl der Scanpunkte des Sensorsystems.

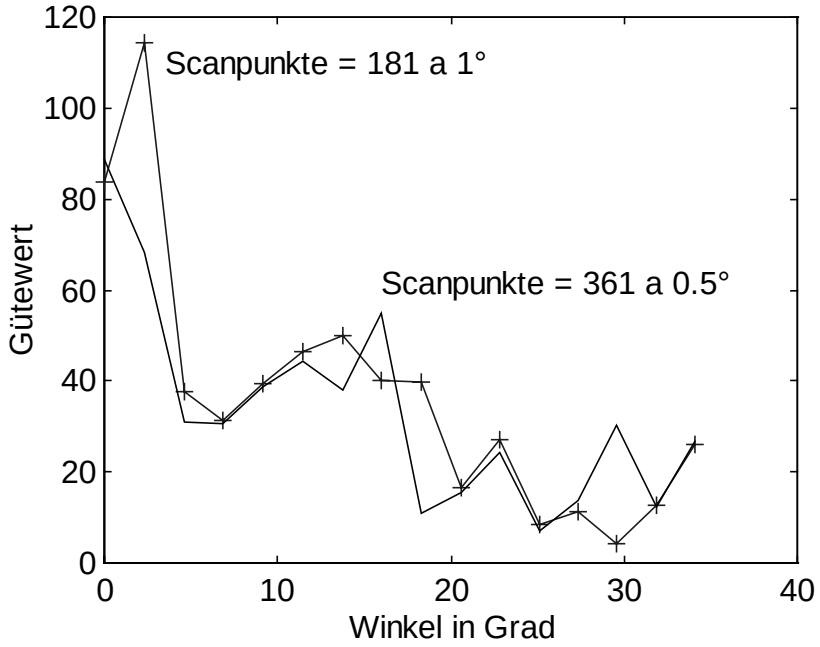

Bild 4.8 Gütewert in Abhängigkeit vom Winkel und der Scanpunktanzahl als Parameter

Die durchgezogene Kurve verdeutlicht das Verhältnis von 0.5°-Schritten mit einer Gesamtscanpunktanzahl von 361. Die mit Kreuzen markierte Kurve zeigt das Verhältnis von 1°-Schritten mit 181 Scanwerten. Der Unterschied ist nicht gravierend, so daß man sagen kann, daß hier auch mit einem billigeren Laser (Sensor) mit nur 181 Scanpunkten auszukommen ist. Ein zweiter Gesichtspunkt der gegen die 361 Scanpunkte spricht, ist die doppelt so große Programmlaufzeit gegenüber der mit 181 Scanpunkten. Das wird aber zum

Nachteil für das Erkennen von kleinen Linien, da hier dann der Unterschied zur Rauschgröße kleiner ist.

Wie oben schon erwähnt wurde, hängt das Erkennen von fast gleichen Linien stark von der Filtergröße ab. Das soll im folgenden noch näher untersucht werden.

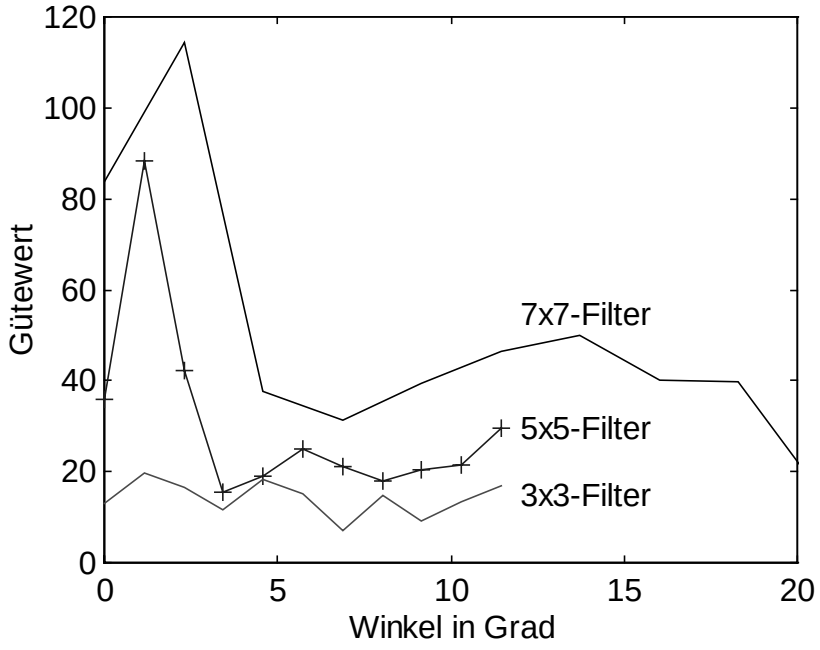

Bild 4.9 Gütewert in Abhängigkeit vom Winkel mit der Filtergröße als Parameter

Das Erkennen von fast gleichen Linien hängt stark von der Filtergröße ab. In Bild 4.9 ist zu erkennen, daß erst bei einem Winkel von 20° zwischen den Linien der Gütewert mit 7x7-Filter dem mit 5x5- oder 3x3-Filter gleicht. Das ist einsehbar, da bei kleinem Winkel und großem Filter, das zweite Maximum verfälscht wird durch die auf Null gesetzten Felder im AF, die bei der Filterrechnung des ersten Maximums entstehen. Es kann sogar passieren, daß Linien, die im AF dicht beieinander liegen, einfach weggefiltert werden, da sie innerhalb der zu Null gesetzten Felder für ein Maximum liegen. Ein 3x3-Filter ist also optimal einzusetzen bei Linien im Bildbereich, die sich nur durch einen kleinen Deltawinkel unterscheiden. Dieser Filter hat natürlich wieder einen Nachteil bei Bildpunkten mit großer Streuung, da Nebenmaxima wieder an Bedeutung gewinnen. Andererseits, tastet man einen quadratischen Raum mit einem Laser ab, kann der Filter nicht groß genug sein, um eine

gute Linienerkennung zu erhalten. Es ist also sehr wichtig, die Filtergröße den Linien im Bildbereich, die erkannt werden sollen, anzupassen.

An allen Grafiken ist zu erkennen, das der Gütewert nicht annähernd Null wird. Dies liegt an der noch zu ungenügenden Auflösung des AF. Würde man anstatt mit *BBSIZE* = 5 cm mit 2 cm rechnen, würden sich die Werte weiterhin verbessern. Aber jede Vergrößerung des AF geht aber zu lasten des Speicherplatzes und der Ausführungsgeschwindigkeit und sollte deshalb je nach Aufgabenstellung mit Bedacht gewählt werden.

Nun wird die minimale Grenzlänge einer Linie im Bildbereich, für die eine Erkennung erfolgt, untersucht. Diese Frage läßt sich unmittelbar aus dem nächsten Bild beantworten. Es wurde eine Streuung von ± 4 cm zugrunde gelegt.

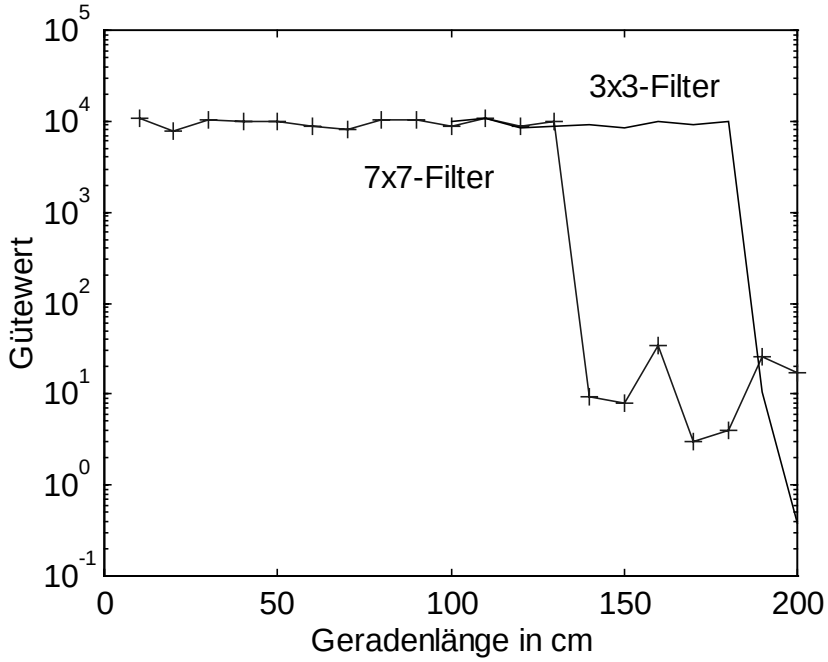

Bild 4.10 Gütewert in Abhängigkeit von der Linienlänge und der Filtergröße als Parameter

Es ist gut zu sehen, daß ein merklicher Unterschied zwischen einem 3x3- und einem 7x7-Filter besteht. Arbeitet man mit einem 7x7-Filter, werden kürzere Linien sicherer erkannt als bei einem kleineren Filter. Das wird klar, wenn man sich überlegt, daß große Filter die Nebenmaxima von großen Linien mehr wegfiltern, also eine größere Umgebung um die

Hauptmaxima zu Null setzen. Somit haben dann Hauptmaxima von kleinen Linien eher eine Chance, erkannt zu werden. Benutzt man jetzt nur einen 3x3-Filter, sind die zurückbleibenden Nebenmaxima einer großen Linie immer noch größer als das Hauptmaximum einer kleinen Linie. Aber auch bei einem 7x7-Filter muß die Linie größer als 140 cm betragen, um erkannt zu werden.

Das Auffinden von kleinen Linien spricht für einen großen Filter. Dieser große Filter trägt dann aber dazu bei, daß Linien mit kleinen Winkeln zueinander nicht erkannt werden. Was für das Erkennen von kleinen Linien gut ist, ist für das Erkennen von wenig unterschiedlichen Linien schlecht. Erkennbar wird, daß es keine optimale Filtergröße gibt. Es kommt darauf an, die Filtergröße geschickt den Linien oder Punktdaten im Bildbereich anzupassen.

Natürlich hängt das Erkennen von kleinen Linien auch entscheidend von der Streuung der Punktdaten ab. Dieser Sachverhalt wird durch das folgende Bild dargestellt.

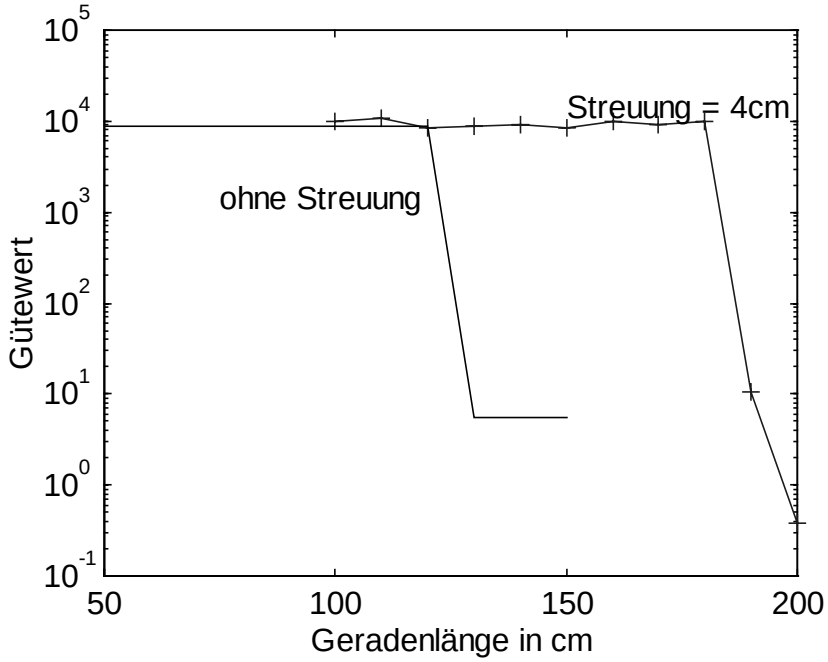

Bild 4.11 Gütewert in Abhängigkeit der Linienlänge mit der Streuung als Parameter

Der Testdurchlauf in Bild 4.11 wurde mit einem 3x3-Filter durchgeführt. Wie zu erwarten war, werden kleine Linien, deren Punktdaten nicht mit einer Streuung versetzt sind, eher erkannt, als verrauschte Punktdaten. Das wird unmittelbar klar, da sich im AF starke Hauptmaxima mit nur relativ kleinen Nebenmaxima bilden. Bei verrauschten Daten verschwimmen die Unterschiede von Haupt- zu Nebenmaxima sehr schnell, die nur mit einer größeren Filterrechnung behoben werden können, natürlich mit all ihren Vor- und Nachteilen.

Als letztes stellt sich die Frage, ob die Liniensegmenterkennung von der jeweiligen Entfernung der Linien vom Ursprung des Lasers abhängt.

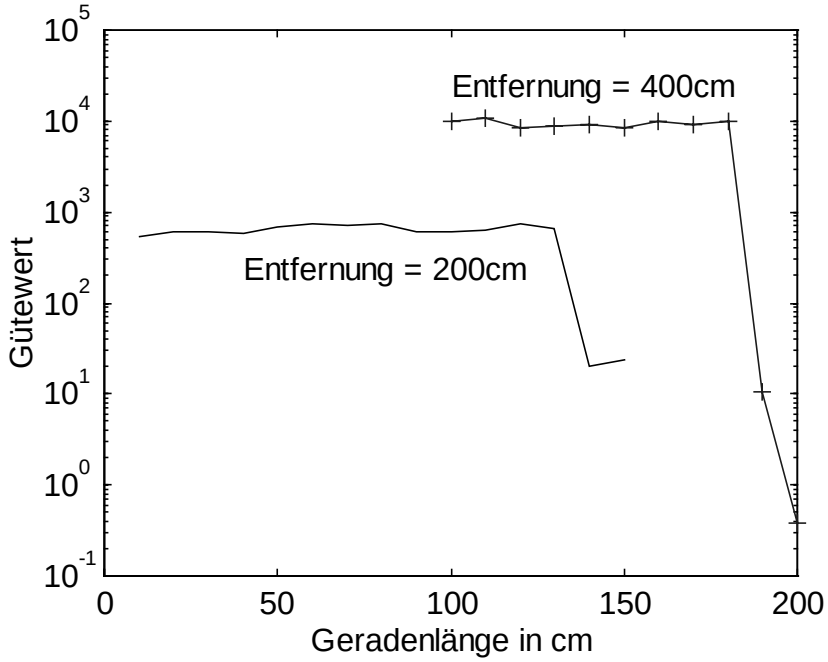

Bild 4.12 Gütewert in Abhängigkeit der Linienlänge mit der Entfernung als Parameter

Wie in Bild 4.12 zu erkennen ist, spielt auch die Entfernung der Linien vom Ursprung des Laserkoordinatensystems im Bildbereich eine große Rolle. Gleich große Linien, die näher am Ursprung liegen, werden früher erkannt als solche mit größerer Entfernung. Dies läßt sich wie folgt begründen: der Scanner umspannt bei kleinerer Entfernung einen kleineren Weg zwischen zwei Scanpunkten, als in größerer Entfernung (Strahlensatz der Mathematik). Dies wird aber nicht weiter zum Problem, da mit der gewichteten Hough-

Transformation die Entfernung der Punktdaten in die Rechnung mit eingeht /Forsberg87/. Forsberg gewichtet die auftretenden Maxima im AF nach ihrer Entfernung vom Koordinatenursprung. Somit wird der zuvor beschriebene Effekt minimiert.

Wie schon am Umfang der Testdaten zu erkennen ist, ist bei der Hough-Transformation das Auswerten der Linienerkennung nach verschiedenen Gesichtspunkten nicht ganz einfach. Der Hough-Transformations-Algorithmus muß an die verschiedenen Anwendungsgebiete angepaßt werden, um gleichermaßen gute Ergebnisse liefern zu können.

4.2 Polyline Segmentierungs-Algorithmus

Beim PSA hat man weniger Parameter für das Einstellen des Algorithmus. Die Güte wird, wie auch bei der Hough-Transformation, in verschiedenen Abhängigkeiten dargestellt. Der PSA ist im Bezug auf die Programmlaufzeit und die Unabhängigkeit der Punktdatenanordnung im Bildbereich besser geeignet zur Liniensegmenterkennung. Beim Messen der Programmlaufzeit für den PSA kamen Werte im Millisekundenbereich heraus, die ich hier nicht mehr darstellen will. Es versteht sich natürlich von selbst, daß, bei Halbierung der Scanpunktanzahl, auch eine Halbierung der Programmlaufzeit erfolgt.

Als einzigen leicht einstellbaren Parameter kann man beim PSA die *SCHWELLE* verändern. Mit dem Schwellwert wird der Wert bezeichnet, der die maximale Distanz eines neuen Punktes zur gerade erzeugten Linie angibt. Wird mit der Distanz dieser Schwellwert überschritten, so bildet dieser Punkt den Beginn einer neuen Linie.

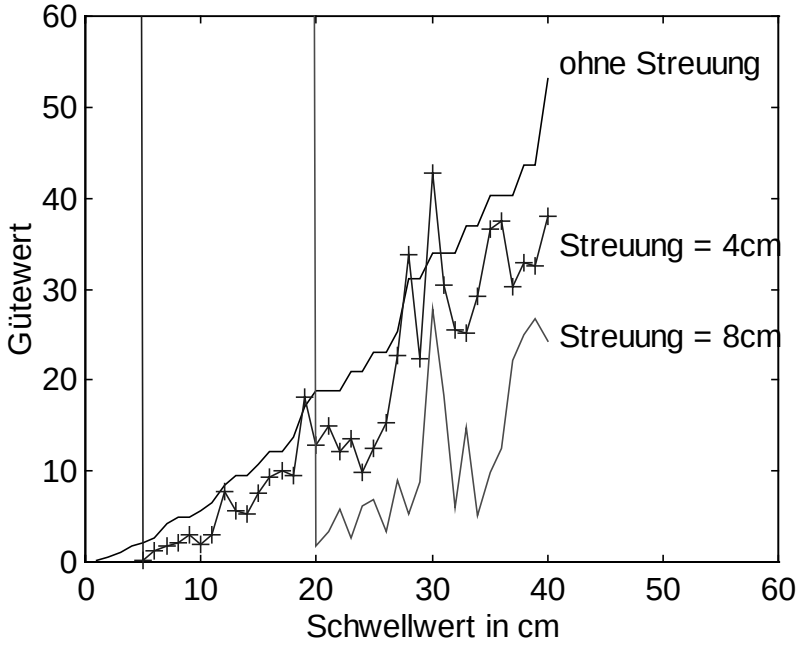

Bild 4.13 Gütewert in Abhängigkeit vom Schwellwert mit der Streuung als Parameter

Bei einem Durchlauf ohne Streuung steigt der Gütewert quadratisch mit der Erhöhung des Schwellwertes. Die Genauigkeit der Linienerkennung verschlechtert sich mit der Erhöhung der Schwelle. Es gehen also bei einem großen Schwellwert noch einige Punktdaten der nächsten Linie in die aktuelle Linie mit ein. Diese wenigen Punktdaten verschlechtern die Hesse-Parameter der aktuellen Linie, und gehen für die Erkennung der nächsten Linie verloren. Je größer die Schwelle gewählt wird, desto mehr falsche Punktdaten gehen mit in die Berechnung der aktuellen Linie ein. Bei größer werdender Streuung ist die Wahrscheinlichkeit größer, daß schon eher ein Punktwert existiert, der den Schwellwert übersteigt, um die Berechnung der aktuellen Linie zu beenden. Dieser Effekt der Streuung ist auch auf die Ungleichverteilung der Pseudo-Zufallszahlen-Funktion zurückzuführen. Somit liegt die Kurve mit einer Streuung von ± 8 cm unter der Kurve mit ± 4 cm Streuung. Daraus leitet sich unmittelbar der unruhige Verlauf der beiden Kurven mit Streuung her.

Viel interessanter ist natürlich die Frage nach dem Anfangspunkt der drei Kurven. Bei allen drei Testdurchläufen beginnt der Schwellwert bei einem Wert von 1. Ein Wert von Null würde keinen Sinn machen. Bei reinen Punktdaten ohne Verrauschung beginnt der Güteverlauf nahe Null, da bei einer Schwelle von 1 sofort jeder Punkt, der nicht zur Linie ge-

hört, erkannt wird und als Startpunkt für die nächste Linie gesetzt wird. Dadurch werden die Hesse-Parameter der Linien nur minimal verfälscht.

Bei einer Streuung von ± 4 cm beginnt der Güteverlauf ungefähr bei 5 cm Schwellwert. Bei einer großen Streuung und einem Schwellwert von 1 oder 2 werden natürlich keine Linien erkannt, bei einer maximale Streuungsweite von 8 cm. Der Punktdatenwert wird mit hoher Wahrscheinlichkeit einen größeren Abstand zum Nächsten haben als 1, 2 oder sogar 3 cm. Die Funktion *Streuung(...)* benutzt zur Generierung der Zufallszahlen die Funktion *rand(...)* aus der C-Bibliothek. Diese Funktion erzeugt Pseudo-Zufallszahlen, die annähernd einer Gleichverteilung folgen. Die Standardabweichung berechnet sich somit zu:

$$\sigma = \frac{b - a}{\sqrt{12}} = \frac{\text{Streuungsweite}}{\sqrt{12}} = \frac{8\,\text{cm}}{\sqrt{12}} = 2.3094\,\text{cm} \qquad (4.1)$$

Überlegt man sich nun weiter, daß wir die Distanz eines Punktes zur aktuellen Linie mit dem Schwellwert vergleichen, stellen wir fest, daß die Standardabweichung in den Wert für die Distanz quadratisch eingeht. Quadrieren wir nun die Standardabweichung, erhalten wir einen Wert von 5.3 cm^2. Dieser Wert steht für den Schwellwert, bei dem bei einer Streuung von ± 4 cm eine Linienerkennung sinnvoll beginnt. Bei einer Streuung von ± 8 cm errechnet sich ein Schwellwert von 21.3 cm^2. Beide Werte stimmen auch in der Grafik relativ gut überein.

Dieser Zusammenhang wird noch deutlicher im folgenden Bild.

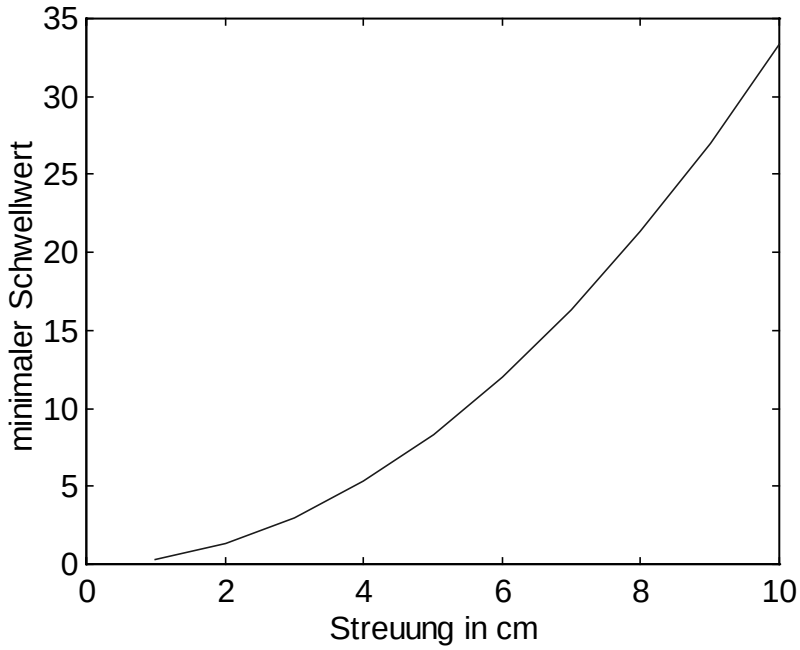

Bild 4.14 minimal geeigneter Schwellwert in Abhängigkeit von der Streuung

Man erkennt hier einen quadratischen Verlauf, da die Standardabweichung der Streuung in den Wert für die Distanz quadratisch eingeht.

Weiter ist nun interessant, ob die Anzahl der Scanpunkte des Lasersensors Auswirkungen auf den Gütewert hat. Bild 4.15 verdeutlicht diesen Zusammenhang.

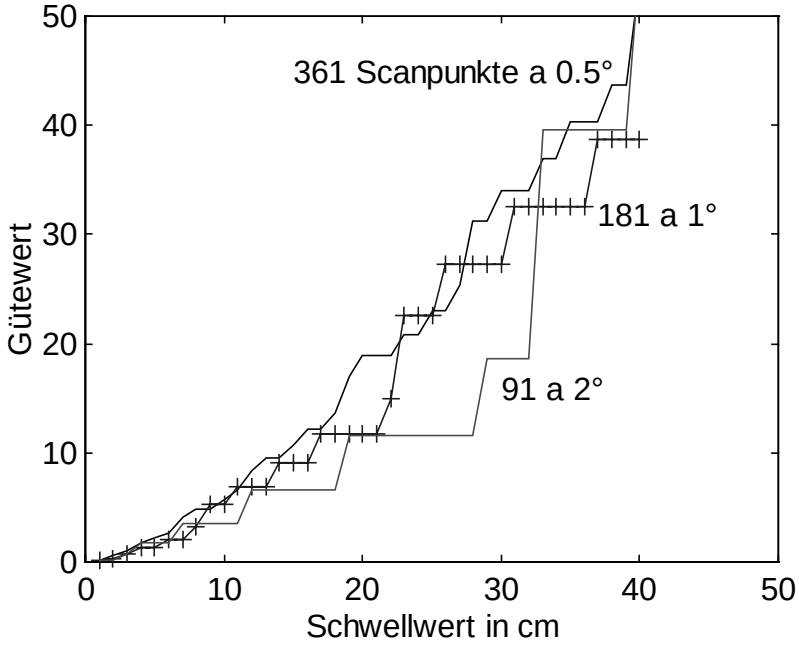

Bild 4.15 Gütewert in Abhängigkeit vom Schwellwert mit der Anzahl der Scanpunkte als Parameter (ohne Streuung)

Man erkennt auch hier wieder nur einen kleinen Unterschied zur Kurve mit 181 Scanpunkten. Die Kurven werden immer unruhiger, da jeder Scanpunkt vom nächsten Scanpunkt einen größeren Abstand haben kann, als bei 361 Scanpunkten. Wenn ein Punkt gerade noch innerhalb der Schwelle liegt, dann kann der nächste Punkt schon wesentlich weiter weg liegen, als der nächste Scanpunkt bei einem Testdurchlauf mit 361 Scanpunkten. Hier stellt sich natürlich die Frage, ob eine Erniedrigung der Scanpunkte bei der geringen Programmlaufzeit überhaupt sinnvoll ist. Bei großer Scanpunktanzahl (≥ 361), wird der Kurvenverlauf fast stetig, und die Wahrscheinlichkeit, eine gute Linienerkennung zu erhalten, nimmt zu. Das kann man sich auch relativ leicht anhand der kurzen Algorithmusbeschreibung im Abschnitt 2.2.1 klarmachen.

Als nächstes interessiert uns nun wieder, wie der Algorithmus auf verschiedene Linienlängen mit der Erkennung reagiert, um nachher einen Vergleich mit der Hough-Transformation zu ziehen. Es wird hier der Parameter *STREUUNG* zu Null gesetzt.

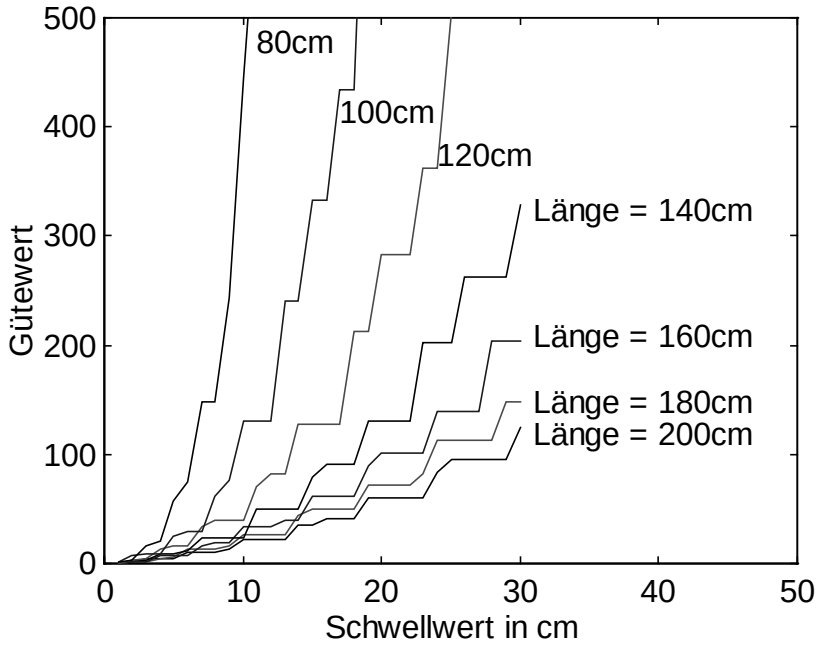

Bild 4.16 Gütewert in Abhängigkeit des Schwellwertes und der Linienlänge als Parameter

Wie schon festgestellt wurde (vgl. Bild 4.13), wird mit der Erhöhung des Schwellwertes der Gütewert näherungsweise quadratisch größer. Der leicht zackige Kurvenverlauf ergibt sich aus der Tatsache, daß wir hier nur mit 181 Scanpunkten arbeiten. Wie in Bild 4.16 gut zu sehen ist, wird mit der Erniedrigung der Linienlänge der zu erkennenden Geraden, der quadratische Verlauf der Güte immer steiler. Es wird deutlich, daß kleinere Linien bei größer werdender Schwelle weniger erkannt werden.

Bevor der PSA einen Punkt zum Startpunkt einer neuen Linie macht, gehen noch eine Zahl von Punkten, eigentlich Punkte der neuen Linie, in die alte Linie mit ein. Je größer die Schwelle gewählt wird, desto mehr Punkte der neuen Linie werden der alten Linie zugeordnet. Dadurch verringert sich die Anzahl der Punkte, die eigentlich zu einer Linie gehört. Dies wird natürlich zum Problem bei kleinen Linien mit wenigen Scanpunkten. Ist die zu erkennende Linie so klein, und die Schwelle dazu so groß, daß alle Punkte in die alte Linie

mit eingehen, wird diese kleine Linie nicht erkannt. Daraus folgt natürlich auch, daß die alte Linie, durch die nicht dazu gehörenden Punkte der kleinen Linie, in den Hesse-Parametern verfälscht wird.

Dieser Zusammenhang wird in Bild 4.16 dargestellt. Bei einer Schwelle von 10, wird eine 80 cm lange Linie nicht mehr ausreichend erkannt. Will man diese kleine Linie noch gut erkennen, so muß man mindestens einen Schwellwert von 4 nehmen. Dem kleinen Schwellwert ist wiederum durch die Streuung der Laserpunktdaten eine minimale Grenze gesetzt. Möchte man dagegen eine 200 cm lange Linie noch gut erkennen, so kann man einen größeren Schwellwert, z.B. 10, nehmen. Es wird also sinnvoll, den Schwellwert beim PSA, je nach Länge der zu erkennenden Linien im Bildbereich, im Bezug auf die Streuung der Punktdaten zu wählen. Sollen vorwiegend Wände in einem Raum erkannt werden, kann man die Schwelle risikolos größer wählen, als bei einer Fehlererkennung von Werkstücken, da in diesem Fall Konturunebenheiten erkannt werden sollen.

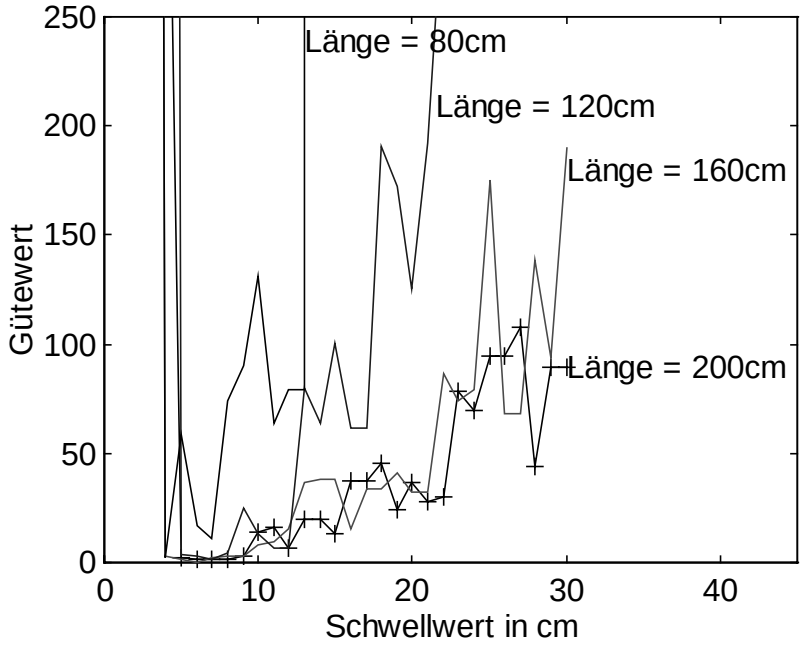

Bild 4.17 Gütewert in Abhängigkeit des Schwellwertes mit der Linienlänge als Parameter

In Bild 4.17 ist der selbe Sachverhalt wie in Bild 4.16 dargestellt, nur mit einer Streuung von ± 4 cm. Der Kurvenverlauf wird wieder unruhiger, durch die verrauschten Punktdaten,

und die Startpunkte der Kurven beginnen nicht bei einem minimalen, sondern bei einem maximalen Gütewert. Das sind alles zuvor beschriebene und bekannte Eigenschaften, die durch die Erhöhung der Streuung auftreten. Man sieht in diesem Bild gut, das es sehr schwer wird, kleine Linien mit 80 cm Länge bei einer Verrauschung der Punktdaten noch hinreichend genau zu erkennen.

Zunächst wollen wir uns nun wieder die Abhängigkeit vom Winkel zwischen zwei Linien anschauen mit verschiedenen Schwellwerten als Parameter. Man kann jetzt schon sagen, daß das Erkennen von verschiedenen Linienlängen und das Erkennen von Linien mit verschiedenen Winkel zwischeneinander im engen Zusammenhang zu einander steht.

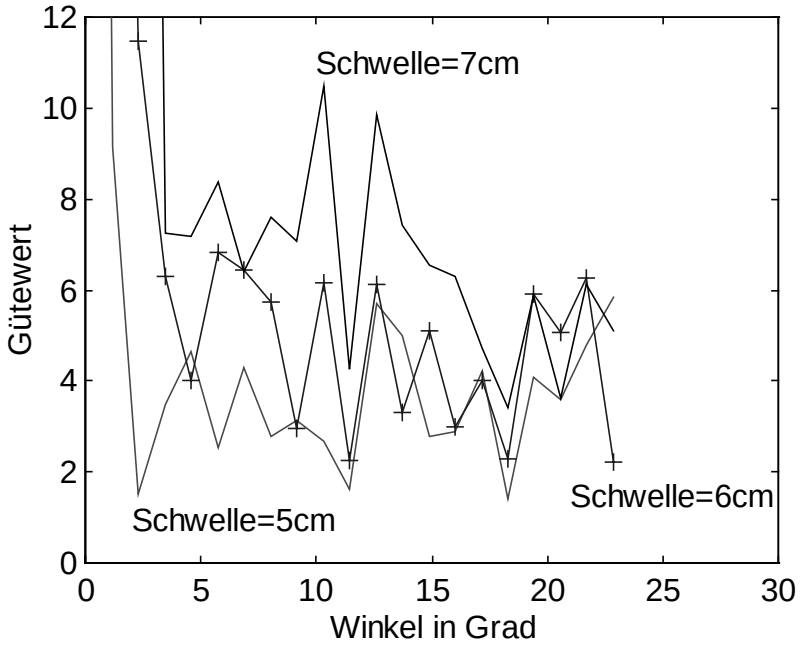

Bild 4.18 Gütewert in Abhängigkeit vom Winkel und der Schwelle als Parameter

Man sieht hier wieder, daß der Gütewert bei höher werdendem Schwellwert erst bei einem größeren Winkel zwischen den Linien abnimmt. Wenn man also durch eine große Streuung die Schwelle groß wählen muß, wird eine Linie, die zu einer anderen Linie nur einen kleinen Winkel hat, schlechter erkannt. Das wird auch wieder unmittelbar klar, da durch die große Schwelle alle Punktdaten der neuen Linie in dem Distanzschwellwert liegen und somit in die alte Linie mit eingehen. Dadurch wird natürlich die Genauigkeit der Hesse-

Parameter der Linie verschlechtert - der Gütewert steigt. Die Größe des Schwellwertes hängt also auch von der Lage der Linien im Raum ab. Will man eine Linienerkennung in einem Bildbereich mit nur kleinen Unterschieden der Hesse-Parameter der Linien durchführen, so sollte man einen Lasersensor mit möglichst geringer Streuung verwenden.

Bei einer festen Streuung und variablem Schwellwert stellt sich im Bezug auf die Genauigkeit der Linienerkennung ein optimales Verhältnis zwischen den beiden her. Das sieht so aus, daß z.B. bei einer Streuung von ± 4 cm ein Schwellwert von nahe 5 die genauesten Werte liefert. Diese Verhältnisse sind im folgenden in Form einer Grafik zusammengestellt.

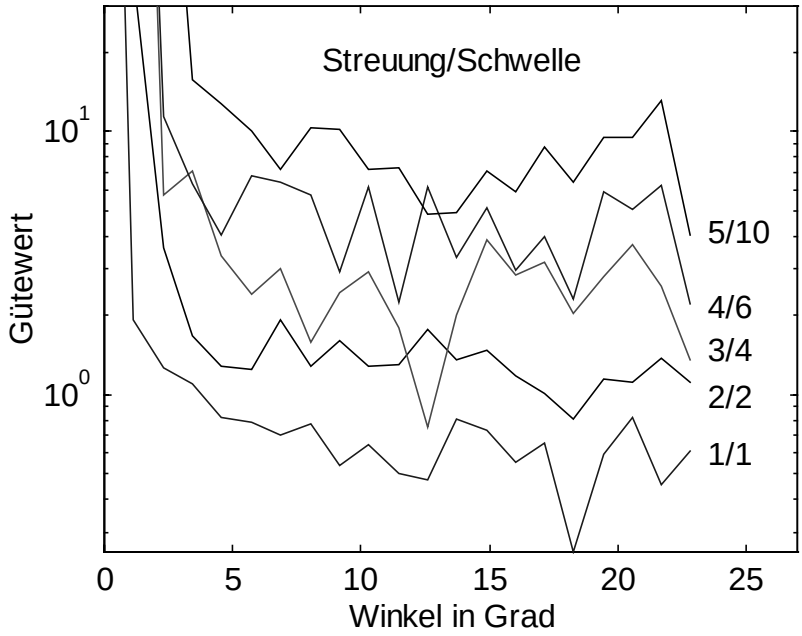

Bild 4.19 Gütewert in Abhängigkeit des Winkels und das Verhältnis von Schwelle zur Streuung als Parameter

Die Grafik verdeutlicht gut die verschiedenen Startwerte bei größer werdender Streuung bzw. Schwelle. Die Güte der Linienerkennung sinkt mit der Erhöhung des Schwellwertes, der sich wiederum an der Größe der Streuung orientiert. Man kann also sagen, wie auch aus den vorhergehenden Erläuterungen hervorgeht, daß die Größe der Streuung der Punktdaten im Bildbereich wesentlich zur Genauigkeit der Linienerkennung beiträgt. Ein noch so

guter Algorithmus zur Linienerkennung kann nicht den Nachteil einer stark verrauschten Datengröße kompensieren.

4.3 Vergleich der beiden Verfahren

Nach den ausführlichen Erläuterungen zu beiden Konturerkennungsverfahren kristallisieren sich einige Vor- und Nachteile deutlich heraus.

Die Hough-Transformation bietet mehr Möglichkeiten zum Einstellen des Algorithmus. Was sicher als Vorteil bezeichnet werden kann, da man hier die Möglichkeit hat, den Algorithmus auf die Linienerkennung im Bildbereich genauer anzupassen. Die Hough-Transformation kann auch weiter zur Kreiserkennung, selbst zur Erkennung von analytisch nicht darstellbaren Kurven, eingesetzt werden /Hesse85/.

Alle anderen Argumente wie Programmlaufzeit oder Fehlerbehandlung sprechen eindeutig für den Polyline Segmentierungs-Algorithmus. Auch der vergleichbare Gütewert liegt beim PSA meist unter dem der Hough-Transformation. Um bei der Hough-Transformation eine vergleichbare Güte zu erreichen, müssen erhöhter Speicherplatzbedarf und größere Programmlaufzeit in Kauf genommen werden. Was beim PSA einige Millisekunden dauert, kann bei der Hough-Transformation schon einige Minuten in Anspruch nehmen.

Da die Hough-Transformation eine Parametertransformation ist, muß jeder Punkt in einen Parameterraum (AF) transformiert werden. Dieses AF muß relativ groß gewählt werden, um eine hohe Güte der Linienerkennung zu gewährleisten. Es muß zum Auffinden der Maxima dann jedes Feld im AF durchsucht werden. Je größer das AF, desto größer wird die Laufzeit. Dazu kommt natürlich auch die Programmlaufzeit der Filterrechnung. Die gesamte Laufzeit wird dann so groß, daß sie für eine Realzeitanwendung nicht mehr in Frage kommt. Hat man zur Linienerkennung viel Zeit und ausreichend Speicherplatz zur Verfügung, so lassen sich mit der Hough-Transformation auch gute Resultate erzielen. Sie ist durchaus für einige Anwendungen besser geeignet, da man sie durch relativ viele Parameter auf die spezielle Anwendung anpassen kann.

Der PSA ist ein sehr schneller Algorithmus zur Linienerkennung. Da er die Daten von aufeinanderfolgenden Punkten auswertet, nutzt er direkt die Informationen der Punktdaten. Er berechnet die gesuchten Hesse-Parameter im Bildbereich und geht nicht über den Umweg einer Parametertransformation. Die Rechenzeit hängt hier nur von der Anzahl der Scanpunkte ab, die natürlich bei der Hough-Transformation noch zu längerer Laufzeit führt. Was bei der Hough-Transformation als Vorteil gewertet wird, kann beim PSA zum Nachteil werden; und zwar die Anzahl der einstellbaren Parameter. Der PSA hat nur den Schwellwert, um den Algorithmus an die Verrauschung der Punktdaten - Streuung des Sensorsystems - anzupassen. Die Qualität seiner Linienerkennung ist dann nur noch mit mathematischer Modifizierung des Algorithmus zu verbessern.

Abschließend kann man eindeutig sagen, daß der PSA in Sachen Realzeitfähigkeit (Programmlaufzeit), Speicherplatzbedarf, Fehleranfälligkeit, Qualität der Linienerkennung und Programmcodegröße (entscheidend bei EPROM-Einsatz) dem Hough-Transformations-Algorithmus vorzuziehen ist.

5 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden zwei Verfahren zur Liniensegmentierung, die Hough-Transformation und der Polyline Segmentierungs-Algorithmus, untersucht. Linienerkennungsverfahren errechnen aus vorhandenen kollinearen Sensorpunktdaten die Parameter einer Linie.

Eine kurze mathematische Vorstellung der Algorithmen, gefolgt von den jeweiligen Korrekturverfahren, bildet die theoretische Einleitung der Arbeit. Die Hough-Transformation nutzt einen Parameterraum, um die Hesse-Parameter einer Linie zu erhalten. Der PSA dagegen verwendet die direkten Punktdaten zur Linienerkennung und garantiert dadurch eine geringere Programmlaufzeit.

Die verwendete Hardware - Lasersensor und Computersystem - spielt eine entscheidende Rolle für die Güte und die Programmlaufzeit der Linienerkennung.

Die Algorithmen wurden in der Programmiersprache C implementiert und dazu eine C-Testumgebung entwickelt. Es können Testdurchläufe mit verschiedenen Parametern gestartet werden, um die Qualität der Linienerkennung in Abhängigkeit von einzelnen Parametern beurteilen zu können. Weiterhin ermöglicht die Umgebung das Verändern von Parametern und Linien im Bildbereich bei laufendem Testprogramm. Diese Testumgebung bildet ein gelungenes Werkzeug zur simulationstechnischen Erprobung von weiteren Liniensegmentierungsverfahren.

Anhand von Korrekturverfahren wurden die Algorithmen weiter optimiert. Mittels eines Filters, der in seiner Größe variabel einzustellen war, konnte die Hough-Transformation noch an Qualität gewinnen.

Um die Algorithmen vergleichen zu können, ist ein Gütekriterium entwickelt worden. Es vergleicht die nominalen mit den errechneten Hesse-Parametern. Mittels der C-Testumgebung wurden dann wichtige Abhängigkeiten untersucht und graphisch dargestellt. Anhand der Kurvenverläufe ließen sich die Vor- und Nachteile der einzelnen Verfahren sehr gut erkennen. Es wurden Zusammenhänge zwischen der Anordnung der Linien im

Bildbereich und verschiedenen Parametern in Bezug auf die Güte der Linienerkennung verdeutlicht.

Dabei kristallisierte sich heraus, daß der PSA für die meisten Anwendungen, besonders im Bereich zur Unterstützung von Navigationsverfahren, den besseren Algorithmus darstellt. Durch seine Realzeitfähigkeit und seinen geringen Programmcode eignet er sich auch für den EPROM-Einsatz auf mobilen Robotersystemen.

Eine weitere Forschungsrichtung ist der Vergleich beider Verfahren mit anderen Algorithmen der Linienerkennung, um einen Algorithmus zu entwickeln, der in der Lage ist, sich mittels weniger Parameter auf verschiedene Linienanordnungen im Bildbereich einzustellen. Mit der Hough-Transformation ist es möglich, auch andere analytisch darstellbare wie auch nicht analytisch darstellbare Kurven zu erkennen.

In Bezug auf mobile Robotersysteme spielt die Umgebungserkennung mittels 2D-(3D-) Laserscanner eine große Rolle. Man kann mit ihr die unveränderlichen wie auch langsam veränderlichen Merkmale der Umgebung, in der sich ein Fahrzeug bewegt, zur Referenzierung heranziehen. Dadurch wird eine permanente Überwachung der Navigation möglich. Weiterhin können mit diesen Verfahren Karten der Umgebung erstellt und im Verlauf ihres Einsatzes verändert oder vergrößert werden.

Linien- oder Kantenerkennungsalgorithmen erlangen in Industrie und Wirtschaft steigende Bedeutung. Mit ihnen können Daten aus verschiedensten Sensoren in analytisch darstellbare Form gebracht werden. Es werden also interessierende Kurven, wie Linien oder Kreise von der großen Datenmenge getrennt und als Parameter dieser Kurve abgelegt. In der Medizin werden Verfahren dieser Art (Hough-Transformation) angewendet, um einzelne Gehirnhälften zu segmentieren und anschließend darzustellen.

Der Einsatz von Linien- oder Konturerkennungsverfahren wird auch in der Zukunft interessant bleiben, da viele Bereiche der Bilddatenverarbeitung und Mustererkennung erst am Anfang ihrer Entwicklung stehen. Mit genauer werdenden Sensoren zur Meßdatenaufnahme sowie schnelleren und zuverlässigeren Hardwarelösungen wird mit der Verbesserung der Liniensegmentierungsverfahren der Weg zu mobilen Robotersystemen im Bereich der Navigation weiter vorangetrieben.

6 Literaturverzeichnis

/Ballard81/ Ballard, D. H., Generalizing the Hough transform to detect arbitrary shapes, Pattern Recognition, 13:111-122, 1981

/Duda72/ Duda, R. O., Hart, P. E., Using the Hough transform to detect lines and curves in pictures, Communications of the ACM, 15(1):11-15, 1972

/Duda73/ Duda, R. O., Hart, P. E., Pattern Classification and Scene Analysis, John Wiley and Sons, New York, 1973

/Ekman/ Ekman, A., Klöör, P., Strömberg, D., Incremental Map-Making in indoor environments, National Defence Research Establishment, Dept. of Information Technology, Linköping, Schweden

/Forsberg87/ Forsberg, J., Larsson, U., Åhman, P., Wernersson, Å., The Hough Transform inside the Feedback Loop of a Mobile Robot, Robotics & Automation, University of Luleå, Schweden, S-951, 1987

/Freund93/ Freund, E., Dierks, F., Roßmann, J., Untersuchungen zum Arbeitsschutz bei Mobilen Robotern und Mehrrobotersystemen, Schriftenreihe der Bundesanstalt für Arbeitsschutz, Forschung Fb 682, Dortmund, 1993

/Gonzalez92/ Gonzalez, J., Stentz, A., Ollero, A., An Iconic Position Estimator for a 2D Laser Range Finder, Field Robotics Center, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 15213 USA, 1992

/Hesse85/ Hesse, H.-J., Konturverfolgung zur Analyse von Grauwertbildern, Dissertation des Fachbereiches Elektrotechnik der Fernuniversität in Hagen, 1985

/Hough62/ Hough, P. V. C., A Method and Means for Recognizing Complex Pattern, U.S., Patent 3.069.654, 1962

/Whyte94/ Durrant-Whyte, H. ,Where Am I ?, A Tutorial on Mobile Vehicle Localization, Industrial Robot, Vol. 21 No. 2, pp. 11-16, MCB University Press, 1994

/Princen94/ Princen, J., Illingworth, J., Kittler, J., Hypothesis Testing: A Framework for Analyzing and Optimizing Hough Transform Performance, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 16, No. 4, April 1994

Anhang

Nachfolgend eine Übersicht der verwendeten Funktionen in alphabetischer Reihenfolge:

Programmquelltext:

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>
#include <string.h>
#include <alloc.h>
#include <mem.h>
#include <time.h>

#define KLEIN           1e-07
#define PI              3.1415926535897323846

extern unsigned _stklen = 51200U;

//////////////////////////////////////////////////////////////
////  WERTE AUS DEM PARAMETERFILE
//////////////////////////////////////////////////////////////
int GR_FAK;             //5, faktor für kleinere anzeige auf screen
int BBSETX;             //110, Ursprung x der graphik, horizontal
int BBSETY;             //110, Ursprung y, vertikal
int RESSETX;            //500, ursprung x der graphik der ges. geraden
int RESSETY;            //460

int WSTEP;              //32, Winkelschritt im transbereich 3-d
int BBSIZE;             //2, quantisierungsfehler bzw. breite
int REICHWEITE;         //Laserreichweite
int MAX_DIST;           //REICHWEITE / BBSIZE
int RMAX;               //MAX_DIST *2
int MIN_PUT_R;          //0, min und max dist-wert für 3d-graphik
int MAX_PUT_R;          //100, 0 ... 500; -250 ... 250, da alles nicht passt
int TBANZSKAL;          //-2
```

```
int FOCUSCOUNT;                 //6, anzahl der auszuwertenden maxima

int MAX_SCAN;                   //361, maximale anzahl d. scanpunkte(180ø) des lasers
float PHI_STEP_SCAN;            //0.5, winkelschrittweite des lasers
int STREUUNG;                   //4, genauigkeit des lasers ist +-4cm
int FILTER_SCHW_R;              //Filterschwellwert,ab dem er arbeitet
int FILTER_SCHW_PHI;

int SCHWELLE;                   //Punktschwellwert f   r den PSA-Algorithmus

#include "studarb.h"

#define ANZ_HESSE_BB            10
#define ANZ_FOCUS               10
static BILD_BEREICH bild_bereich[362];          //r und phi
static TRANS_BEREICH hesse_bb[ANZ_HESSE_BB];    //r,phi,schnittpunktanzahl
static double hesse_bb_winkel[ANZ_HESSE_BB-1];  //winkel zw. den linien
static TRANS_BEREICH focus[ANZ_FOCUS];          //r, phi und schnittpunktanzahl

////////////////////////////////////////
////    FUNKTIONEN
////////////////////////////////////////
void GetParameter( char *pfname)
{
        FILE *paramfile;
        char string1[254],string2[254];
        paramfile = (FILE*)fopen( pfname, "r+t");
        if( paramfile==NULL )                           // datei existiert nicht
        {
                printf( "Parameterfile existiert nicht.");
                exit(1);
        }
        fseek( paramfile, 0L, SEEK_SET);                // Dateizeiger an den Anfang
        fscanf( paramfile,"%s", &string1[0]);           //für Grafikdarstellung
        fscanf( paramfile,"%s%d%s", &string1[0], &GR_FAK, &string2[0]);
```

```
        fscanf( paramfile,"%s%d%s", &string1[0], &BBSETX, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &BBSETY, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &RESSETX, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &RESSETY, &string2[0]);
        fscanf( paramfile,"%s", &string1[0]);                    //für Berechnung im AF
        fscanf( paramfile,"%s%d%s", &string1[0], &WSTEP, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &BBSIZE, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &REICHWEITE, &string2[0]);
        MAX_DIST = (int)(REICHWEITE/BBSIZE);
        RMAX = MAX_DIST * 2;
        fscanf( paramfile,"%s%d%s", &string1[0], &MIN_PUT_R, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &MAX_PUT_R, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &TBANZSKAL, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &FOCUSCOUNT, &string2[0]);
        fscanf( paramfile,"%s", &string1[0]);                    //zur Generierung des Lasers
        fscanf( paramfile,"%s%d%s", &string1[0], &MAX_SCAN, &string2[0]);
        fscanf( paramfile,"%s%f%s", &string1[0], &PHI_STEP_SCAN, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &STREUUNG, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &FILTER_SCHW_R, &string2[0]);
        fscanf( paramfile,"%s%d%s", &string1[0], &FILTER_SCHW_PHI, &string2[0]);
        fscanf( paramfile,"%s", &string1[0]);                    //für PSA Algorithmus
        fscanf( paramfile,"%s%d%s", &string1[0], &SCHWELLE, &string2[0]);
        fclose( paramfile);
}

void SetParameter( char *pfname, char *xyfname, char *str)
{
        int i=0,von,in,bis,zeile=0;
        float wert;
        static char parameter[20],*s,string1[255],string2[255],fname[15];
        static char buffer[40][255];
        FILE *pfile;
        s = strtok( &str[0],"=");
        sscanf( s, "%s", &parameter[0]);
        s = strtok( NULL,"[");
```

```
sscanf( s, "%d", &von );
s = strtok( NULL,"]");
sscanf( s, "%d", &in );
s = strtok( NULL,"\n");
sscanf( s, "%d", &bis );

if( strstr(&parameter[0],"PUNKT")==NULL )
        strcpy( &fname[0], &pfname[0]);                  //Parameter soll verändert werden
else if( strstr(&parameter[0],"PUNKT")!=NULL )
        strcpy( &fname[0], &xyfname[0]);                 //Pkt-Wert soll verändert werden

if( von<=bis )                                           //Test noch nicht abgeschlosen
{
        pfile = (FILE*)fopen( fname, "rt");
        if( pfile==NULL) { printf("Datei %s",fname);exit(1); };
        fseek( pfile, 0L, SEEK_SET);                     // Dateizeiger an den Anfang
        do                                          //Zeile für den Parameter in "param.c" finden
        {
                zeile++;
                fscanf( pfile,"%s%f%s", &string1[0], &wert, &string2[0]);
        }while( strcmpi(&string1[0],&parameter[0])!=0 );
        fseek( pfile, 0L, SEEK_SET);
        while( feof(pfile)==0 )                          //alles in einen Puffer
                fgets( &buffer[i++][0],254,pfile);
        fclose( pfile);

        strset( &buffer[zeile-1][0],' ');
        sprintf( &buffer[zeile-1][0],"%s  %d  %s\n",
                &parameter[0], von, &string2[0]);

        pfile = (FILE*)fopen( fname, "wt");               //neu zum schreiben öffnen
        if( pfile==NULL) { printf("Datei %s",fname);exit(1); };
        i=0;
        while( strlen( &buffer[i][0])!=0 )               //wenn Puffer nicht leer
                fprintf( pfile,"%s", &buffer[i++][0]);
```

```
                fclose( pfile);

                if( von+in<=bis )                       //neuen teststr generieren
                        sprintf( &str[0],"%s=%d[%d]%d", &parameter[0], von+in, in, bis);
                else
                        strset( &str[0],'\0');          //Testende signalisieren
        }
}

void ConvertXYToPhiR( BILD_BEREICH *bb, TRANS_BEREICH *hessebb,
        double *hessebbphi, char *infname, char *outfname)
{
        FILE *infile,*outfile;
        char str[255],s1[100],s2[100],*pstr;
        int i,j,anz=0,hesseanz=0,ierr;
        double x0,y0,x1,y1,teta,p;
        double r,phi,h_phi=0,h_r;

        infile = fopen( infname, "rt");
        if( infile==NULL) { printf("Datei %s",infname);exit(1); };
        outfile = fopen( outfname, "wt");
        if( outfile==NULL) { printf("Datei %s",outfname);exit(1); };
        fgets( str,254,infile);                         //eine leerzeile scannen
        fgets( str,254,infile);
        sscanf( str,"%s%d%s", &s1[0], &i, &s2[0]);
        fgets( str,254,infile);
        sscanf( str,"%s%d%s", &s1[0], &j, &s2[0]);
        x0 = (double)i;
        y0 = (double)j;
        h_r = (double)sqrt( x0*x0 + y0*y0 );
        h_phi = (double)( acos(x0/h_r)*180/PI );        //in Grad
        bb[anz].r = (double)h_r;
        bb[anz].phi = (double)h_phi;                    //winkel in Grad
        fprintf( outfile,"%.10lf %.10lf", h_phi, h_r);  //in Grad
        while( feof(infile)==0 )
```

```
{
        pstr = fgets( str,254,infile);
        ierr = sscanf( pstr,"%s%d%s", &s1[0], &i, &s2[0]);
        if( ierr!=3 )
                break;
        fgets( str,254,infile);
        sscanf( str,"%s%d%s", &s1[0], &j, &s2[0]);
        x1 = (double)i;
        y1 = (double)j;
        r = (double)sqrt( x1*x1 + y1*y1 );
        phi = (double)( acos(x1/r)*180/PI );                    //Winkel des neuen Teilstücks in °
        for( i=0;h_phi<phi;i++,h_phi+=PHI_STEP_SCAN )
                ;       //i=Anzahl der Winkelschritte bis zum nächsten Wendepunkt
        h_phi = h_phi-i*PHI_STEP_SCAN;                          //alten Zähler wiederherstellen in °
        if( x0!=x1 && y0!=y1 )                          //Winkel d Hessegeraden im Bogenmaß
                if( (x0>x1 && y0<y1) || (x0<x1 && y0>y1) )
                        teta = (double)(PI/2 - atan( fabs(y1-y0)/fabs(x1-x0) ));
                else
                        teta = (double)(PI/2 + atan( fabs(y1-y0)/fabs(x1-x0) ));
        else if( x1==x0 && x1>0 )                               //rechte Seite
                teta = 0;
        else if( x1==x0 && x1<0 )                               //linke Seite
                teta = (double)(PI);
        else if( y0==y1 )                                       //waagerechte Wand
                teta = (double)(PI/2);
        else
                { printf("Teta der Hessegeraden falsch.\n");exit(1); }
        p = (double)(x0*cos(teta) + y0*sin(teta));              //Abstand der Hessegeraden
        hessebb[hesseanz].r = (double)p;                //Hessegeradenparam f. Gütekriterium
        hessebb[hesseanz].phi = (double)(teta*180/PI);          //in Grad
        hessebb[hesseanz].anz = 1;
        if( hessebb[hesseanz].phi<0 )
        {                                                       //Winkel nur von 0..<180°
                hessebb[hesseanz].phi = hessebb[hesseanz].phi+180;
                hessebb[hesseanz].r = -hessebb[hesseanz].r;
```

```
                }else if( hessebb[hesseanz].phi>=180 )
                {
                        hessebb[hesseanz].phi = hessebb[hesseanz].phi-180;
                        hessebb[hesseanz].r = -hessebb[hesseanz].r;
                }
                if( hesseanz>=1 )                       //nullte und erste Hessegerade errechnet
                {                                               //Winkel zwischen beiden errechnen
                        hessebbphi[hesseanz-1] =
                            fabs(hessebb[hesseanz-1].phi-hessebb[hesseanz].phi);      //in Grad
                }
                hesseanz++;
                for( j=1;j<i;j++ )                      //Quantisierungsstücke durchlaufen
                {
                        anz++;
                        h_phi = h_phi + PHI_STEP_SCAN;          //in Grad
                        bb[anz].phi = (double)h_phi;
                        bb[anz].r = (double)p/cos(teta-h_phi*PI/180);         //im Bogenmaß
                        fprintf( outfile,"\n%.10lf %.10lf", bb[anz].phi, bb[anz].r);
                }
                x0 = x1;
                y0 = y1;
        }
        fclose( infile);
        fclose( outfile);
}

void SetStreuung( BILD_BEREICH *bb, double streuung)
{
        int i;
        double plus_minus;
        randomize( );
        for( i=0;i<MAX_SCAN;i++ )
        {
                plus_minus = ((double)random(10000))/10000-0.5;
                plus_minus = 2*(plus_minus)*streuung;
```

```
                bb[i].r += plus_minus;
        }
}

void PutBildGraph( BILD_BEREICH *bb)
{
        int i;
        double x,y;
        setcolor( WHITE);
        line( BBSETX, BBSETY, BBSETX, BBSETY-20*GR_FAK-10);
        for( i=1;i<=GR_FAK;i++ )
                line( BBSETX-2, BBSETY-20*i, BBSETX+2, BBSETY-20*i);
        outtextxy( BBSETX+2, BBSETY-20*GR_FAK-10, "Y");
        line( BBSETX-20*GR_FAK-10, BBSETY, BBSETX+20*GR_FAK+10, BBSETY);
        for( i=-GR_FAK;i<=GR_FAK;i++ )
                line( BBSETX+20*i, BBSETY-2, BBSETX+20*i, BBSETY+2);
        outtextxy( BBSETX+20*GR_FAK+10+2, BBSETY, "X");

        for( i=0;i<=MAX_SCAN;i++)
        {                                                       //phi rein im Bogenmaß
                x = (double)( BBSETX + bb[i].r*cos(bb[i].phi*PI/180)/GR_FAK );
                y = (double)( BBSETY - bb[i].r*sin(bb[i].phi*PI/180)/GR_FAK );
                putpixel( (int)x, (int)y, WHITE);
        }
};

void CalcAkumulatorFeld( BILD_BEREICH *bb, AKUMULATOR_FELD af )
{
        int i,l,k,m;
        double r,phi;
        for( i=0;i<MAX_SCAN;i++)                                //Alle ScanPunkte durchlaufen
        {
                for( l=0,phi=0;l<WSTEP;l++,phi+=PI/WSTEP )      //Waagerechte im AF
                {                                               //phi hier im Bogenmaß
                        r = (double)( bb[i].r * cos(bb[i].phi*PI/180-phi) );
```

```
                        if( (r>=-MAX_DIST*BBSIZE) && (r<=MAX_DIST*BBSIZE) )
                        {                                       //r muß im AF sein
                                for( k=0;k<RMAX;k++)            //Senkrechte im AF
                                {                               //Q-stufen des Radius durchlaufen
                                        m = (-MAX_DIST+(k+1))*BBSIZE;
                                        if( r <= m )
                                        {
                                                af[l][k] ++;
                                                break;
                                        }
                                }
                        }
                }
        }
}

void WriteAFFile( AKUMULATOR_FELD af, char *fname)
{
        int i,j;
        FILE *f;
        f = fopen( fname, "wt");
        if( f==NULL ) { printf("Datei %s",fname);exit(1); };
        for( i=RMAX-1;i>=0;i-- )
        {
                fprintf( f, "%3d %4d", i, (-MAX_DIST+(i+1))*BBSIZE );
                for( j=0;j<WSTEP;j++ )
                        fprintf( f,"%2d ", af[j][i] );
                fprintf( f,"\n" );
        }
        fclose( f);
}

void GetMaxima( AKUMULATOR_FELD af, TRANS_BEREICH *focus)
{
        int i,k,l,m=0,i_count;
```

```
AKUMULATOR_FELD *af_ptr;
af_ptr = (AKUMULATOR_FELD*)( (&af)[0][0] );
for( i=MAX_SCAN+1;i>=1;i-- )                    //maximal MAX_SCAN Kurven schneiden
{
        i_count = 0;            //erkennt, ob mehr als ein max mit Höhe i gefunden wird
        for( k=0;k<RMAX;k++ )
        {
                for( l=0;l<WSTEP;l++)
                {
                        if( af[l][k] == i )             //Maximum gefunden
                        {
                                i_count++;              //zählt Maxima mit gleicher Höhe i
                                Filter( *af_ptr, &focus[m], "r_phi", l, k);
                                if( focus[m].phi<0 )
                                {                       //Winkel nur von 0..<180°
                                        focus[m].phi = focus[m].phi+180;
                                        focus[m].r = -focus[m].r;
                                }else if( focus[m].phi>=180 )
                                {
                                        focus[m].phi = focus[m].phi-180;
                                        focus[m].r = -focus[m].r;
                                }
                                m++;
                                if( (m>=FOCUSCOUNT) && (i_count==1) )
                                {       //alles abbrechen, mehr Maxima sind nicht sinnvoll
                                        l = WSTEP;
                                        k = RMAX;
                                        i = 0;
                                        focus[m].phi = 0;
                                        focus[m].r = 0;
                                        focus[m].anz = 0;
                                }
                        }
                }
        }
```

```
        }
}

void Filter( AKUMULATOR_FELD af, TRANS_BEREICH *focus, char *was,int l,int k)
{
        static int i,j,r_von,r_bis,phi_von,phi_bis,f_g_r,f_g_phi;
        int max_ueber_alles,k_merker,i_merker;
        double h_zaehler,zaehler=0,nenner=0;
        double max_anz_j,zaehler_ueber_j,nenner_ueber_j;
        double max_anz_i,zaehler_ueber_i,nenner_ueber_i;

        if( strcmpi(was,"r")==0 )
                f_g_r = (int)floor(FILTER_SCHW_R/BBSIZE);       //Filtergröße r
        else if( strcmpi(was,"phi")==0 )
                f_g_phi = (int)floor(WSTEP/FILTER_SCHW_PHI);
        else if( (strcmpi(was,"r_phi")==0) || (strcmpi(was,"phi_r")==0) )
        {
                f_g_phi = (int)floor(WSTEP/FILTER_SCHW_PHI);
                f_g_r = (int)floor(FILTER_SCHW_R/BBSIZE);
        }
//Distancefiltergrenzen setzen
        for( ;; )
        {
                if( (k-f_g_r>=0) && (k+f_g_r<=RMAX) )           //Filtergrenzen setzen
                {                               //an den Grenzen von r im AF machen wir den Filter kleiner
                        r_von = -f_g_r;
                        r_bis = f_g_r;
                        break;
                }else
                        f_g_r--;
        }
//Winklefiltergrenzen setzen
        phi_von = -f_g_phi;                     //brauchen Winkel im AF nicht zu begrenzen, da
        phi_bis = f_g_phi;                      //mit analytischer Fortsetzung programmiert ist
//komplexe Filterrechnung für r
```

```
k_merker = k;
zaehler_ueber_j = nenner_ueber_j = 0;
for( j=phi_von;j<=phi_bis;j++ )
{
        zaehler = nenner = max_anz_j = 0;
        for( i=r_von;i<=r_bis;i++ )
        {
                k = k_merker;
                i_merker = i;
                if( (l+j>=0) && (l+j<WSTEP) )
                        h_zaehler = ((-MAX_DIST+(k+1+i))*BBSIZE -
                            (double)(BBSIZE)/2);
                else                                    //bei analyt. Fortsetzung von phi
                {
                        k = RMAX-k-1;           //muß Vorzeichen von r gedreht werden
                        i = -i;
                        h_zaehler = -((-MAX_DIST+(k+1+i))*BBSIZE -
                            (double)(BBSIZE)/2);
                }
                zaehler += h_zaehler * af[(l+j+WSTEP)%WSTEP][k+i];
                nenner += af[(l+j+WSTEP)%WSTEP][k+i];
                if( af[(l+j+WSTEP)%WSTEP][k+i]>max_anz_j )
                                                        //Maximum suchen von Spalte j
                        max_anz_j = af[(l+j+WSTEP)%WSTEP][k+i];
                i = i_merker;
        }
        if( (phi_von!=phi_bis) && (r_von!=r_bis) )
        {                                       //wenn über r und phi gemittelt werden soll
                if( nenner!=0 )
                {                       //nenner==zaehler==0 heißt keine Aufaddierung
                        zaehler_ueber_j += (double)( zaehler/nenner * max_anz_j );
                        nenner_ueber_j += max_anz_j;
                }
        }else
        {                       //nur ueber phi oder r gemittelt, äußere Schleife, oder gar nicht
```

```
                zaehler_ueber_j += zaehler;
                nenner_ueber_j += nenner;
            }
        }
        if( nenner_ueber_j!=0 )
            focus->r = (double)( zaehler_ueber_j/nenner_ueber_j );
//komplexe Filterrechnung für phi
        k = k_merker;
        zaehler_ueber_i = nenner_ueber_i = 0;
        for( i=r_von;i<=r_bis;i++ )
        {
            i_merker = i;
            zaehler = nenner = max_anz_i = 0;
            for( j=phi_von;j<=phi_bis;j++ )
            {
                k = k_merker;
                if( (l+j<0) || (l+j>=WSTEP) )
                {                                   //wenn außerhalb vom AF (wie oben)
                    k = RMAX-k-1;           //es muß Vorzeichen von r gedreht werden
                    i = -i;                 //Gewichtungsreihenfolge im AF umdrehen
                }
                h_zaehler = ((l+j)*((double)(180))/WSTEP);          //in Grad
                zaehler += h_zaehler * af[(l+j+WSTEP)%WSTEP][k+i];
                nenner += af[(l+j+WSTEP)%WSTEP][k+i];
                if( af[(l+j+WSTEP)%WSTEP][k+i]>max_anz_i )
                                                    //Max suchen von Spalte i
                    max_anz_i = af[(l+j+WSTEP)%WSTEP][k+i];
                i = i_merker;               //alte Gewichtungsreihenfolge wiederherstellen
            }
            if( (r_von!=r_bis) && (phi_von!=phi_bis) )
            {                                   //wenn über r und phi gemittelt werden soll
                if( nenner!=0 )
                {                               //wie bei r Aufaddierung von 0
                    zaehler_ueber_i += (double)( zaehler/nenner * max_anz_i );
                    nenner_ueber_i += max_anz_i;
```

```
                    }
                }else
                {                       //nur über r oder phi gemittelt (äußere Schleife) oder gar nicht
                    zaehler_ueber_i += zaehler;
                    nenner_ueber_i += nenner;
                }
            }
            if( nenner_ueber_i!=0 )
                focus->phi = (double)( zaehler_ueber_i/nenner_ueber_i );
//f_g_phi mal f_g_r Quadrat Null setzen und nur das Max stehen lassen
            k = k_merker;
            max_ueber_alles = af[l][k];                         //Maximum retten
            for( i=r_von;i<=r_bis;i++ )
            {
                for( j=phi_von;j<=phi_bis;j++ )
                {
                    k = k_merker;
                    if( (l+j<0) || (l+j>=WSTEP) )
                                                    //bei analytischer Fortsetzung von phi
                        k = RMAX-k-1;               //es muß Vorzeichen von r gedreht werden
                    af[(l+j+WSTEP)%WSTEP][k+i] = 0;
                }
            }
            k = k_merker;
            af[l][k] = max_ueber_alles;
            focus->anz = (int)af[l][k];
}

void PutHesseValue( int xa, int ya, TRANS_BEREICH hesse_value[])
{
            int i,anz=0;
            char str[30];

            while( hesse_value[anz].anz!=0 )
                anz++;
```

```
        for( i=0;i<anz;i++ )
        {
                if( hesse_value[i].anz == 1 )                   //beim Aufruf mit bb-Geraden
                        setcolor( WHITE);
                else if( hesse_value[i].anz>0 )
                        setcolor( i+2);             //beim Aufruf mit resultierenden Hesse Geraden
                sprintf( str, "%i", i+1);
                outtextxy( xa, ya+i*30, str);
                setcolor( 15);
                sprintf( str, "Phi: %.2fø", hesse_value[i].phi);
                outtextxy( xa+10, ya+i*30, str);
                sprintf( str, "R  : %.2f", hesse_value[i].r);
                outtextxy( xa+10, ya+10+i*30, str);
                sprintf( str, "Anz: %d", (int)(hesse_value[i].anz));
                outtextxy( xa+10, ya+20+i*30, str);
        }
}

void PutResultGraph( TRANS_BEREICH focus[])
{
        int i,rx0,ry0,rx1,ry1,anz=0;
        line( RESSETX, RESSETY, RESSETX, RESSETY-20*GR_FAK-10);
        for( i=1;i<=GR_FAK;i++ )
                line( RESSETX-2, RESSETY-20*i, RESSETX+2, RESSETY-20*i);
        outtextxy( RESSETX+2, RESSETY-20*GR_FAK-10, "Y");
        line( RESSETX-20*GR_FAK-10, RESSETY,
                RESSETX+20*GR_FAK+10, RESSETY);
        for( i=-GR_FAK;i<=GR_FAK;i++ )
                line( RESSETX+20*i, RESSETY-2, RESSETX+20*i, RESSETY+2);
        outtextxy( RESSETX+20*GR_FAK+10+2, RESSETY, "X");
        while( focus[anz].anz!=0 )
                anz++;
        for( i=0;i<anz;i++ )
        {
                if( (focus[i].phi<45) || (focus[i].phi>135) )           //in Grad abfragen
```

```
                {
                        ry0 = (int)( 0 );
                        rx0 = (int)( (focus[i].r/cos(focus[i].phi*PI/180)-
                                0*sin(focus[i].phi*PI/180)/cos(focus[i].phi*PI/180)
                                )/GR_FAK);
                        ry1 = (int)( 20*GR_FAK );
                        rx1 = (int)( (focus[i].r/cos(focus[i].phi*PI/180)-
                                500*sin(focus[i].phi*PI/180)/cos(focus[i].phi*PI/180)
                                )/GR_FAK);
                }else
                {
                        rx0 = (int)( -20*GR_FAK );
                        ry0 = (int)( (focus[i].r/sin(focus[i].phi*PI/180)-
                                -500*cos(focus[i].phi*PI/180)/sin(focus[i].phi*PI/180)
                                )/GR_FAK );                     //alle phi's in Bogenmaß
                        rx1 = (int)( 20*GR_FAK );
                        ry1 = (int)( (focus[i].r/sin(focus[i].phi*PI/180)-
                                500*cos(focus[i].phi*PI/180)/sin(focus[i].phi*PI/180)
                                )/GR_FAK );
                }
                setcolor( i+2);
                setlinestyle( 0, 1, 1);
                line( RESSETX+rx0, RESSETY-ry0, RESSETX+rx1, RESSETY-ry1);
        }
}

void PutSurface( H_AKUMULATOR_FELD h_af, int dist)
{
        int l;
        //für Gitter waagerechte Linien malen
        moveto( (int)(h_af[0][0].phi), (int)(h_af[0][0].r) );
        for( l=1;l<WSTEP;l++ )
                lineto( (int)(h_af[l][0].phi), (int)(h_af[l][0].r) );

        if( dist==0 )
```

```
            outtextxy( h_af[0][0].phi-40, h_af[0][0].r, "-500");
if( dist==ceil(0.25*MAX_DIST) )
            outtextxy( h_af[0][0].phi-40, h_af[0][0].r, "-375");
if( dist==ceil(0.5*MAX_DIST) )
            outtextxy( h_af[0][0].phi-40, h_af[0][0].r, "-250");
if( dist==ceil(0.75*MAX_DIST) )
            outtextxy( h_af[0][0].phi-40, h_af[0][0].r, "-125");
if( dist==MAX_DIST )
            outtextxy( h_af[0][0].phi-40, h_af[0][0].r, "   0");
if( dist==ceil(1.25*MAX_DIST) )
            outtextxy( h_af[0][0].phi-40, h_af[0][0].r, " 125");
if( dist==ceil(1.5*MAX_DIST) )
            outtextxy( h_af[0][0].phi-40, h_af[0][0].r, " 250");
if( dist==ceil(1.75*MAX_DIST) )
            outtextxy( h_af[0][0].phi-40, h_af[0][0].r, " 375");
if( dist==2*MAX_DIST-1 )
            outtextxy( h_af[0][0].phi-40, h_af[0][0].r, " 500");

for( l=0;l<WSTEP;l++ )                              //Für Gitter senkrechte Linien malen
{
            line( (int)(h_af[l][0].phi), (int)(h_af[l][0].r),
                        (int)(h_af[l][1].phi), (int)(h_af[l][1].r) );
            if( dist==MIN_PUT_R+1 )
            {                                                   // nur beim ersten Mal ausgeben
                        if( l==0 )
                                    outtextxy( h_af[l][0].phi-5, h_af[l][0].r+15, "0°");
                        if( l==ceil(0.5*WSTEP) )
                                    outtextxy( h_af[l][0].phi-5, h_af[l][0].r+15, "90°");
                        if( l==WSTEP-1 )
                                    outtextxy( h_af[l][0].phi-5, h_af[l][0].r+15, "<180°");
            }
            h_af[l][0].r = h_af[l][1].r;          //1 auf 0 laden, da in Rot() neuer Wert auf 1
            h_af[l][0].phi = h_af[l][1].phi;
            h_af[l][0].anz = h_af[l][1].anz;
}
```

```
}

void Rot( AKUMULATOR_FELD af, float alpha, float beta, float gamma,
        float sx, float sy, float sz, int xt, int yt, int zt)
{
        H_AKUMULATOR_FELD       *h_af_ptr;      //für 2 aufeinanderfolgende r's zum malen
        float a,b,c,d,e,f,g,h,i,al,be,ga;
        int k,l,m=0,spx,spy,spz;

        h_af_ptr = (H_AKUMULATOR_FELD *)farcalloc( 1L,
                (unsigned long)sizeof(H_AKUMULATOR_FELD) );
        if( h_af_ptr==0 )
        {
                printf("Malloc konnte h_af-feld nicht öffnen.\n");
                exit(1);
        }

        al=(float)gamma*PI/180;
        be=(float)beta*PI/180;
        ga=(float)alpha*PI/180;
        a=(float)cos(al)*cos(be);
        b=(float)sin(al)*cos(be);
        c=(float)-sin(be);
        d=(float)-sin(al)*cos(ga)+cos(al)*sin(be)*sin(ga);
        e=(float)cos(al)*cos(ga)+sin(al)*sin(be)*sin(ga);
        f=(float)cos(be)*sin(ga);
        g=(float)sin(al)*sin(ga)+cos(al)*sin(be)*cos(ga);
        h=(float)-cos(al)*sin(ga)+sin(al)*sin(be)*cos(ga);
        i=(float)cos(be)*cos(ga);
        for( k=0;k<RMAX;k++ )
        {
                if( k>=MIN_PUT_R  &&  k<=MAX_PUT_R  )
                {
                        for( l=0;l<WSTEP;l++)
                        {
```

```
                                spx = (short)ceil( sx*( a*PI/WSTEP*l
                                        + d*(k-MAX_DIST)*BBSIZE
                                        + g*af[l][k]*TBANZSKAL) + xt);
                                spy = (short)ceil( sy*( b*PI/WSTEP*l
                                        + e*(k-MAX_DIST)*BBSIZE
                                        + h*af[l][k]*TBANZSKAL) + yt);
                                spz = (short)ceil( sz*( c*PI/WSTEP*l
                                        + f*(k-MAX_DIST)*BBSIZE
                                        + i*af[l][k]*TBANZSKAL) + zt);
                                (*h_af_ptr)[l][m].phi = (short)spx;
                                (*h_af_ptr)[l][m].r   = (short)spy;
                                (*h_af_ptr)[l][m].anz = (short)spz;
                        }
                        if( k>MIN_PUT_R && k<MAX_PUT_R )
                                PutSurface( *h_af_ptr, k);
                        else
                                m=1;
                }
        }
        farfree( h_af_ptr);
}

double Guetekriterium( TRANS_BEREICH *bb, TRANS_BEREICH *tb)
{
        int bb_anz=0,tb_anz=0,i,j;
        static double r,phi,r_d,phi_d,r_quadrat,phi_quadrat;
        static double r_plus_d,r_minus_d,phi_plus_d,phi_minus_d;
        static double r_plus_d1,r_minus_d1,phi_plus_d1,phi_minus_d1;
        static double jjj;
        while( bb[bb_anz].anz!=0 )
                bb_anz++;
        while( tb[tb_anz].anz!=0 )
                tb_anz++;
        if( bb_anz>tb_anz )                     //eine Gerade weniger gefunden als vorgegeben
                tb_anz = bb_anz;                //um Durchlauf zu gewährleisten ->Fehler steigt
```

```
for( i=0;i<bb_anz; )
{
        for( j=i;j<tb_anz;j++ )
        {                                   //Maxima finden zwischen phi=0 und phi<=180
                r_plus_d = tb[j].r+r_d;
                r_minus_d = tb[j].r-r_d;
                phi_plus_d = tb[j].phi+phi_d;
                phi_minus_d = tb[j].phi-phi_d;

                if( bb[i].phi<2*180/WSTEP )         //Maximum in der Nähe von phi=0
                {
                        r_plus_d1 = -tb[j].r+r_d;
                        r_minus_d1 = -tb[j].r-r_d;
                        phi_plus_d1 = tb[j].phi-180+phi_d;
                        phi_minus_d1 = tb[j].phi-180-phi_d;
                }else if( bb[i].phi>(WSTEP-2)*180/WSTEP )
                {                                   //Maximum in der Nähe von phi=180
                        r_plus_d1 = -tb[j].r+r_d;
                        r_minus_d1 = -tb[j].r-r_d;
                        phi_plus_d1 = tb[j].phi+180+phi_d;
                        phi_minus_d1 = tb[j].phi+180-phi_d;
                }else                               //hat keine Wirkung
                {
                        r_plus_d1 = tb[j].r+r_d;
                        r_minus_d1 = tb[j].r-r_d;
                        phi_plus_d1 = tb[j].phi+phi_d;
                        phi_minus_d1 = tb[j].phi-phi_d;
                }
                if(  (  (bb[i].r<r_plus_d && bb[i].r>r_minus_d)  &&
                        (bb[i].phi<phi_plus_d && bb[i].phi>phi_minus_d)
                    ) || (
                        (bb[i].r<r_plus_d1 && bb[i].r>r_minus_d1)  &&
                        (bb[i].phi<phi_plus_d1 && bb[i].phi>phi_minus_d1)
                    )
                 )
```

```
                {
                        r = tb[i].r;
                        phi = tb[i].phi;                    //alle Winkel in Grad
                        tb[i].r = tb[j].r;
                        tb[i].phi = tb[j].phi;
                        tb[j].r = r;
                        tb[j].phi = phi;
                        j = tb_anz;
                        i++;                                //nächstes Feld, Hessepaar im BB
                        r_d = -1;
                        phi_d = -0.1;                       //in Grad
                }
        }
        r_d += 1;
        phi_d += 0.1;                                       //in Grad
}
jjj = 0;                                                    //bei mehreren Testläufen hintereinander
for( i=0;i<bb_anz;i++ )
{
        bb[i].phi = bb[i].phi * PI / 180;                   //im Bogenmaß, wegen Gütekriterium
        tb[i].phi = tb[i].phi * PI / 180;
        if( pow((double)(bb[i].r-tb[i].r),(double)2) <
                pow((double)(bb[i].r+tb[i].r),(double)2) )
        {                                                   //normale Differenz irgentwo im Feld
                r_quadrat = (double)pow((double)(bb[i].r-tb[i].r),(double)2);
                phi_quadrat = (double)pow((double)((bb[i].phi -
                        tb[i].phi)*bb[i].r),(double)2);
        }else
        {                                                   //neg und pos Dist. -> Winkeldifferenz um PI
                r_quadrat = (double)pow((double)(bb[i].r+tb[i].r),(double)2);
                if( bb[i].r>0 )                             //Original hat pos Dist
                {
                        if( bb[i].phi<PI/WSTEP )
                        //re Wand, im kleinsten Winkelschritt bb.phi um >0, tb.phi um <PI
                                phi_quadrat =  (double)pow((double)((bb[i].phi+
```

```
                                        tb[i].phi-PI)*bb[i].r),(double)2);
                        else if( bb[i].phi>(WSTEP-1)*PI/WSTEP )
                        //linke Wand, im größten Winkelschritt bb.phi um <PI, tb.phi um >0
                                phi_quadrat = (double)pow((double)((bb[i].phi -
                                        PI+tb[i].phi)*bb[i].r),(double)2);
                    }else if( bb[i].r<0 )                   //Original hat neg Dist
                        if( bb[i].phi>(WSTEP-1)*PI/WSTEP )
                        //Rechte wand, bb.phi um <PI, tb.phi um >0
                                phi_quadrat = (double)pow((double)((bb[i].phi -
                                        PI+tb[i].phi)*bb[i].r),(double)2);
                        else if( bb[i].phi<PI/WSTEP )
                        //linke Wand, bb.phi um >0, tb.phi um<PI
                                phi_quadrat =  (double)pow((double)((bb[i].phi+
                                        tb[i].phi-PI)*bb[i].r),(double)2);
                }

                jjj += (double)( r_quadrat + phi_quadrat );         //Winkel in Grad

                tb[i].phi = tb[i].phi * 180 / PI;
                bb[i].phi = bb[i].phi * 180 / PI;                   //in Grad
        }
        return jjj;
}

void PutGueteGraph( int xa, int ya, char *gfname, char *algorithmus, int anz_dl)
{
        int i=1,ch,von,in,bis,faktor;
        double guete,guete_kl=1000;
        static char titel[255],*s,parameter[20],guete_str[20];
        FILE *guete_file;
        guete_file = (FILE*)fopen( gfname,"rt");
        if( guete_file==NULL) { printf("Datei guete_kr.c");exit(1); };
        fgets( &titel[0], 254, guete_file);
        s = strtok( &titel[0],"=");
        sscanf( s, "%s", &parameter[0]);
```

```
s = strtok( NULL,"[");
sscanf( s, "%d", &von );
s = strtok( NULL,"]");
sscanf( s, "%d", &in );
s = strtok( NULL,"\n");
sscanf( s, "%d", &bis );

ch = getchar();
ch = ch;
cleardevice();
setcolor( WHITE);
sprintf( &titel[0],"%s-Gütekriterium in Abhängigkeit von %s: %d..(%d)..%d",
        algorithmus, &parameter[0], von, in, bis);
outtextxy( xa, ya+50, &titel[0]);
line( xa, ya, xa, 20);                                  //Güteachse
outtextxy( xa, 10, "J");
if( anz_dl<20 )
        faktor = 25;                                    //bei kleiner Anzahl größere Breite
else
        faktor = 5;                                 //bei vielen Werten kleinerer Zwischenraum
line( xa, ya, xa+((bis-von)/in+2)*faktor, ya);          //Parameterachse
outtextxy( xa+((bis-von)/in+2)*faktor+20, ya+12, &parameter[0]);
while( feof(guete_file)==0 )
{                                                       //alles in einen Puffer
        fgets( &titel[0], 254, guete_file);
        sscanf( &titel[0],"%lf", &guete );              //Güte holen
        if( ya-guete*5<=20 )
                guete = ((double)ya-20)/5;              //Begrenzung oben, für Grafik
        setcolor( LIGHTGREEN);
        line( xa+i*faktor, ya, xa+i*faktor, ya-(int)(guete*5));
        if( guete<guete_kl )
                guete_kl = guete;
        setcolor( WHITE);
        line( xa+i*faktor, ya, xa+i*faktor, ya+2);      //Striche auf Achse
        if( anz_dl<20 )
```

```
        {
                sprintf( &parameter[0],"%d", von+(i-1)*in);
                outtextxy( xa+i*faktor-8, ya+12, &parameter[0]);     //Achseinteilung
        }else
        {
                if( (i-1)%5==0 )                        //alle 5 Werte eine Achsbeschriftung
                {
                        line( xa+i*faktor, ya+2, xa+i*faktor, ya+4);
                        sprintf( &parameter[0],"%d", von+(i-1)*in);
                        outtextxy( xa+i*faktor-8, ya+12, &parameter[0]);
                }
        }
        i++;
}
setlinestyle( DOTTED_LINE, 1, 1);
line( (int)(xa-2), (int)(ya-guete_kl*5),
        (int)(xa+((bis-von)/in+2)*faktor), (int)(ya-guete_kl*5) );
sprintf( guete_str,"%.2lf", guete_kl );
outtextxy( (int)(xa+((bis-von)/in+2)*faktor+5),
         (int)(ya-guete_kl*5-5), guete_str);
line( (int)(xa-2), (int)(ya-guete_kl*10*5),
        (int)(xa+((bis-von)/in+2)*faktor), (int)(ya-guete_kl*10*5) );
sprintf( guete_str,"%.2lf", guete_kl*10 );
outtextxy( (int)(xa+((bis-von)/in+2)*faktor+5),
         (int)(ya-guete_kl*10*5-5), guete_str);
line( (int)(xa-2), (int)(ya-guete_kl*20*5),
        (int)(xa+((bis-von)/in+2)*faktor), (int)(ya-guete_kl*20*5) );
sprintf( guete_str,"%.2lf", guete_kl*20 );
outtextxy( (int)(xa+((bis-von)/in+2)*faktor+5),
         (int)(ya-guete_kl*20*5-5), guete_str);
line( (int)(xa-2), (int)(ya-guete_kl*100*5),
        (int)(xa+((bis-von)/in+2)*faktor), (int)(ya-guete_kl*100*5) );
sprintf( guete_str,"%.2lf", guete_kl*100 );
outtextxy( (int)(xa+((bis-von)/in+2)*faktor+5),
         (int)(ya-guete_kl*100*5-5), guete_str);
```

```
        setlinestyle( SOLID_LINE, 1, 1);
        fclose( guete_file);
}

void CalcPSA( BILD_BEREICH *bb, TRANS_BEREICH *focus)
{
        int ii,nn=1,kk=1,focuscount=0;
        static double a,b,c,d,e,h,i,j,k;
        static double phi,phi1,phi2,r,r1,r2,dist1,dist2;

        a = b = c = d = e = 0;
        for( ii=0;ii<=MAX_SCAN;ii++,kk++ )
        {
                a += (double)pow( (bb[ii].r*cos(bb[ii].phi*PI/180)), (double)2 );
                b += (double)pow( (bb[ii].r*sin(bb[ii].phi*PI/180)), (double)2 );
                c += (double)( bb[ii].r*bb[ii].r*
                        cos(bb[ii].phi*PI/180)*sin(bb[ii].phi*PI/180) );
                d += (double)( bb[ii].r * cos(bb[ii].phi*PI/180) );
                e += (double)( bb[ii].r * sin(bb[ii].phi*PI/180) );
                if( kk>2 )                      //erst beim dritten Punkt pro Linie beginnt der PSA
                {
                        h = (double)(a - d*d/kk);
                        i = (double)(b - e*e/kk);
                        j = (double)(c - d*e/kk);
                        k = (double)0.5*(h - i);
                        if( k>0 )
                        {
                                phi1 = (double)0.5*( atan(j/k) + PI );          //in Bogenmaß
                                phi2 = (double)0.5*( atan(j/k) - PI );
                        }else if( k<0 )
                        {
                                phi1 = (double)0.5*( atan(j/k) + PI + PI);    //in Bogenmaß
                                phi2 = (double)0.5*( atan(j/k) + PI - PI);
                        }else
                        { printf("PSA, incorrect k use.\n");exit(1); }
```

```
dist1 = (double)1/(kk-1)*( (double)pow( cos(phi1),(double)2 )*h +
        (double)pow( sin(phi1),(double)2 )*i +
        (double)2*sin(phi1)*cos(phi1)*j );
dist2 = (double)1/(kk-1)*( (double)pow( cos(phi2),(double)2 )*h +
        (double)pow( sin(phi2),(double)2 )*i +
        (double)2*sin(phi2)*cos(phi2)*j );
if( (dist1<SCHWELLE) && (nn+kk<MAX_SCAN) )
{       //Dist liegt innerhalb der Schwelle -> Punkt auf der Geraden
        r1 = (double)1/kk*( cos(phi1)*d + sin(phi1)*e );
        r2 = (double)1/kk*( cos(phi2)*d + sin(phi2)*e );
        if( r1>0 )                  //pro hinzukommenden Pkt. neues r, neues phi
        {                           //Werte werden immer genauer (schrittweise)
                r = r1;
                phi = phi1;
        }else if( r2>0 )
        {
                r = r2;
                phi = phi2;
        }else
        { printf("PSA, incorrect r use.\n");exit(1); }
}else
{
        focus[focuscount].r = r;
        focus[focuscount].phi = phi*180/PI;   //in Grad abspeichern
        focus[focuscount].anz = 2;          //für die Ausgabe-void
        if( focus[focuscount].phi<0 )
        {                                   //Winkel nur von 0..<180°
                focus[focuscount].phi += 180;
                focus[focuscount].r = -focus[focuscount].r;
        }else if( focus[focuscount].phi>=180 )
        {
                focus[focuscount].phi -= 180;
                focus[focuscount].r = -focus[focuscount].r;
        }
        focuscount++;
```

```
                                    if( focuscount==ANZ_FOCUS-1 )
                                                                    //zuviele falsche Geraden gefunden
                                        nn = MAX_SCAN;              //Algorithmus abbrechen
                                    if( nn+kk < MAX_SCAN-1 )        //neue Linie
                                    {
                                        nn += kk;
                                        kk = 0;                     //kk erhält ersten Punkt der new line
                                        a = b = c = d = e = 0;
                                    }else
                                    {
                                        focus[focuscount].r = 0;
                                        focus[focuscount].phi = 0;
                                        focus[focuscount].anz = 0;
                                        break;                      //fertig
                                    }
                                }
                            }
                        }
                    }

void main( int argc, char *argv[])                  // 1. Parameter gibt Rechenalgorithmus an
{                                                   // HOUGH oder PSA
        int gd,gm,guete_file_flag=0;                // 2. Parameter holt Testparameter
        int ch,setx,sety,anz_durchlauf=0;
        int anz_hesse_bb_winkel;
        double guete_krit;
        static char param_str[40],param[20],wert[5];
        char *s,p_fname[15],g_fname[15],t_fname[15],w1_fname[15],w2_fname[15];
        time_t t1,t2,laufzeit;
        FILE *guete_file,*p_file,*g_file,*t_file,*w1_file,*w2_file;
        AKUMULATOR_FELD *akumulator_feld_ptr;

        detectgraph( &gd, &gm);
        initgraph( &gd, &gm, "");
```

```
setx=(int)ceil( getmaxx( )/2-50 );
sety=(int)ceil( getmaxy( )/2-25);
do
{
        anz_hesse_bb_winkel = 0;
        if( (argv[1]!=NULL) && (argv[2]!=NULL) )
        {                                                   //testparameter ist angegeben
                if( guete_file_flag==0 )    //File zur Auswertung des Gütekriteriums
                {
                        guete_file_flag = 1;                //soll im Testlauf nur einmal öffnen
                        guete_file = (FILE*)fopen( "hp_guete.dat","wt");
                        if( guete_file==NULL)
                                { printf("Datei guete_kr.dat");exit(1); };
                        fprintf( guete_file,"%s", strupr(argv[2]) );
                        //Parameter und Startwert merken
                        strncpy( &param_str[0], strupr(argv[2]), 40);
                        s = strtok( &param_str[0],"=");
                        sscanf( s, "%s", &param[0]);
                        s = strtok( NULL,"[");
                        sscanf( s, "%s", &wert[0]);
                        //Parameter extra im Matlab-file schreiben
                        sprintf( &p_fname[0],"%c%c%c%c%c_p.dat",
                                param[0],param[1],param[2],param[3],param[4]);
                        p_file = (FILE*)fopen( p_fname,"wt");
                        if( p_file==NULL) { printf("Datei %s",
                                &p_fname[0]);exit(1); };
                        //Guetewert extra im Matlab-file schreiben
                        sprintf( &g_fname[0],"%c%c%c%c%c_g.dat",
                                param[0],param[1],param[2],param[3],param[4]);
                        g_file = (FILE*)fopen( g_fname,"wt");
                        if( g_file==NULL)
                                { printf("Datei %s", &g_fname[0]);exit(1); };
                        //Laufzeitwert extra in Matlab-file schreiben
                        sprintf( &t_fname[0],"%c%c%c%c%c_t.dat",
                                param[0],param[1],param[2],param[3],param[4]);
```

```
                t_file = (FILE*)fopen( t_fname,"wt");
                if( t_file==NULL)
                        {printf("Datei %s",&t_fname[0]);exit(1);};
                //Erster Winkel extra in Matlab-file schreiben
                sprintf( &w1_fname[0],"%c%c%c%c%c_w1.dat",
                        param[0],param[1],param[2],param[3],param[4]);
                w1_file = (FILE*)fopen( w1_fname,"wt");
                if( w1_file==NULL)
                        { printf("Datei %s", &w1_fname[0]);exit(1); };
                //Zweiten Winkel extra in Matlab-file schreiben
                sprintf( &w2_fname[0],"%c%c%c%c%c_w2.dat",
                        param[0],param[1],param[2],param[3],param[4]);
                w2_file = (FILE*)fopen( w2_fname,"wt");
                if( w2_file==NULL)
                        { printf("Datei %s", &w2_fname[0]);exit(1); };
        }
        //Parameterwert herausfiltern und in extra Matlab-file schreiben
        strncpy( &param_str[0], strupr(argv[2]), 40);
        s = strtok( &param_str[0],"=");
        s = strtok( NULL,"[");
        fprintf( p_file," %s", s);
        strncpy( &param_str[0], strupr(argv[2]), 40);
        SetParameter( "hp_param.c","hp_bbxy.c", strupr(argv[2]) );
}
akumulator_feld_ptr = (AKUMULATOR_FELD*)farcalloc
        (1L,(unsigned long)sizeof(AKUMULATOR_FELD));
if( akumulator_feld_ptr==0 )
{
        printf("Farmalloc konnte huge-feld nicht öffnen.\n");
        exit(1);
}
GetParameter( "hp_param.c");
ConvertXYToPhiR( &bild_bereich[0],&hesse_bb[0],&hesse_bb_winkel[0],
        "hp_bbxy.c","hp_phir.dat");
SetStreuung( &bild_bereich[0], STREUUNG);
```

```
if( strcmpi( argv[1],"HOUGH")==0 )
{
        t1 = time( NULL);
        CalcAkumulatorFeld( &bild_bereich[0], *akumulator_feld_ptr);
        WriteAFFile( *akumulator_feld_ptr, "hp_af_1.dat");
        GetMaxima( *akumulator_feld_ptr, &focus[0]);
        t2 = time( NULL);
        laufzeit = difftime( t2, t1);
        WriteAFFile( *akumulator_feld_ptr, "hp_af_2.dat");
        cleardevice( );
        PutBildGraph( &bild_bereich[0]);
        PutHesseValue( 0, 130, &hesse_bb[0]);
        Rot( *akumulator_feld_ptr,250,-0.4,0.1,50,1.2,1,setx-60,sety,0);
}else if( strcmpi( argv[1],"PSA")==0 )
{
        cleardevice( );
        PutBildGraph( &bild_bereich[0]);
        PutHesseValue( 0, 130, &hesse_bb[0]);
        t1 = time( NULL);
        CalcPSA( &bild_bereich[0], &focus[0]);
        t2 = time( NULL);
        laufzeit = difftime( t2, t1);
}else
{
        printf("No Algorithm is present.\n");
        exit(1);
}
PutHesseValue( 530, 0, &focus[0]);
PutResultGraph( &focus[0]);
if( guete_file_flag == 1 )
{
        guete_krit = Guetekriterium( &hesse_bb[0], &focus[0]);
        fprintf( guete_file,"\n%.10lf %ld", guete_krit, laufzeit);
        fprintf( g_file," %.10lf", guete_krit);
        fprintf( t_file," %ld", laufzeit);
```

```
                    fprintf( w1_file," %lf", hesse_bb_winkel[0]);
                    fprintf( w2_file," %lf", hesse_bb_winkel[1]);
                    anz_durchlauf++;
            }
            farfree( akumulator_feld_ptr);
        }while( strlen(argv[2])!=0 );
        if( guete_file_flag == 1 )
        {
            fclose( guete_file);
            fclose( p_file);
            fclose( g_file);
            fclose( t_file);
            fclose( w1_file);
            fclose( w2_file);
            PutGueteGraph( 20, 400, "hp_guete.dat", strupr(argv[1]), anz_durchlauf);
        }
        ch = getchar();
        closegraph();
}
```