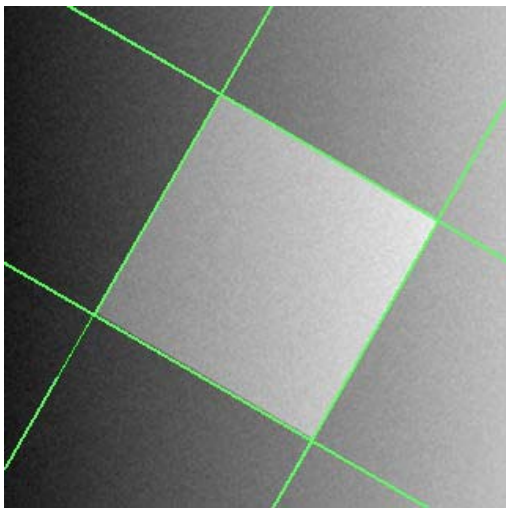
Algorithmische Anwendungen – Projekt 2006

Dokumentation

Hough – Transformation



Volker Schaefer 11038847
Adalbert Zipser 11037149

Index

Motivation

In einem digitalisierten auf dem PC befindlichen Bild sollen Kanten gefunden werden die später in dem Bild nachgezeichnet werden können. Bestandteil dieser Dokumentation sind die mathematischen Hintergründe und die Vorbereitungen der Bilder.
Im Technischen Bereich findet die Hough - Tranformation viele Anwendungen. So können z.B. Roboter mittels der Hough Transformation Gegenstände identifizieren und zu einem bestimmten Platz bringen (gemeint ist hier z.B. ein Roboter der in Büroräumen Papiereimer entleeren kann). Aber nicht nur Roboter werden evtl. mit der Hough Transformation ausgestattet. Auch Maschinen die unterbrochene Leiterbahnen auf Platinen erkennen sollen, nutzen dafür die Hough Transformation. In einer Realzeitumgebung können bestimmte Kontroll- bzw. Überwachungs - Aufgaben automatisiert durchgeführt werden, so lässt sich automatisch die Form und damit später auch die Anzahl bestimmter Werkstücke errechnen / herausfinden, zum Beispiel in der Endkontrolle vor dem verschließen der Verpackung.

Der Ursprung der Hough Transformation geht auf P.V.C Hough zurück, der seine Idee patentieren ließ [1].

In den nun folgenden Kapitel wird die Hough Transformation mathematisch eingeführt. Die weiteren Kapitel beschäftigen sich mit der Vorbereitung der Bilder und der Anwendung der Hough Transformation.

Mathematische Grundlagen

So wie man es von der Mathematik gewohnt ist eine Gerade zu beschreiben, könnte man auch in einem Bild eine Gerade beschreiben; mit:

$$\mathbf{y = ax + b}$$

Beachten muß man dabei aber, daß das Koordinatensystem in einem Bild seinen Ursprung oben links hat.
Ein erster Ansatz eine Gerade im Bild zu identifizieren könnte dazu führen, das wir jeden Bildpixel abfragen, in eine Liste eintragen, und andere in der Nähe befindliche Punkte, den bereits gefundenen Punkten zuordnen. Nur erstens wie sollte die Datendarstellung der Linie dann festgehalten werden ? Außerdem würden wir dann jeden Punkt mit den bereits gefundenen Punkten vergleichen müssen, ob dieser Punkt noch auf der Bahn einer bereits gefundenen Linie liegen würde. Ein weiteres Problem besteht darin, festzustellen ob die Linie eine positive Steigung oder eine negative Steigung hat. Und wenn zwei Geraden sich schneiden, zu welcher Linie gehört dann der Punkt ? Viele unlösbare Probleme würden entstehen.
Einen Teil des oben erwähnten Ansatzes können wir dennoch übernehmen: wir tasten das Bild Pixel für Pixel ab. Dann haben wir die X – Position und die Y – Position eines Bildpunkts. Was uns fehlt, ist die Steigung (a) und der Y – Achsenabschnitt (b). Folglich formen wir die Gleichung nach einem der beiden Paramter um und erhalten:

$$\mathbf{b = y - ax}$$

In einem weiteren Schritt würde man dann nur noch viele verschiedene Werte für eine Steigung einsetzen. Abbildung 1 [1] verdeutlicht das Ergebnis.

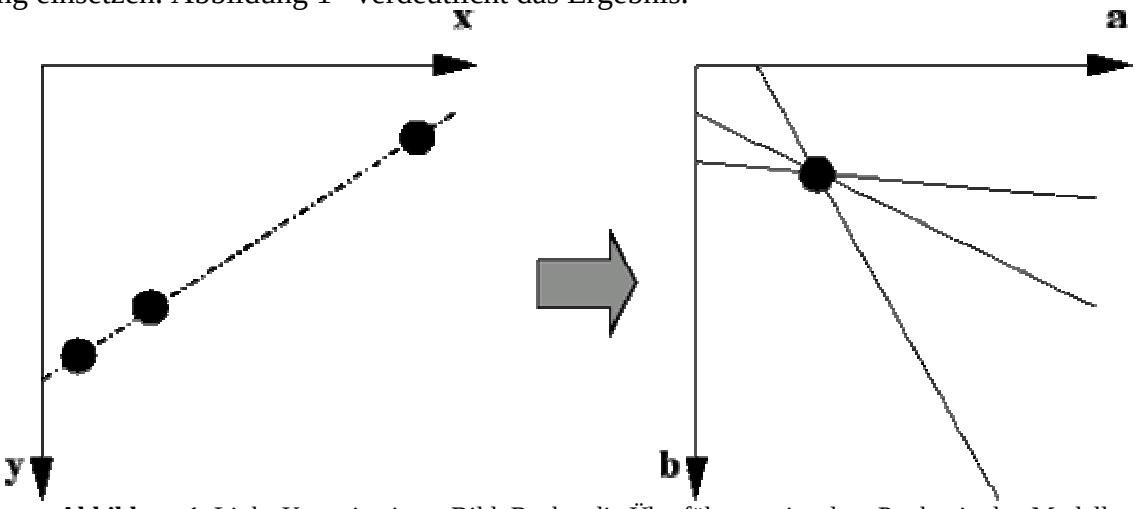


Abbildung 1: Links Kante in einem Bild. Rechts die Überführung einzelner Punkte in den Modellraum

Alle drei Punkte im linken Bild (in Abbildung 1) ergeben die drei Linien im rechten Bild, die nach der Formel **b = y – ax** berechnet wurden und denen eine Reihe von Werten für die Steigung (a) vorgegeben wurde. Der Schnittpunkt der drei Linien im rechten Bild enthält auf der b – Achse den y – Achsenabschnitt aller drei Punkte aus dem linken Bild, sowie auf der a – Achse die gesuchte Steigung (a). Könnten wir in Abbildung 1 die realen Werte für a und b an dem rechten Koordinatensystem ablesen, könnten wir die Linie im linken Bild nachzeichnen.

[1] http://w3studi.informatik.uni-stuttgart.de/%7Eboehmts/onlineversion/HtmlDateien/img155.png

Wichtig zu erwähnen ist hier noch, daß die Umstellung der Formel von **y = ax + b** (Koordinaten- System) in die Parameterform **b = y – ax** eine Transformation in einen neuen Raum bedeutet, nämlich in den Modellraum.
Leider können wir diesen Ansatz so nicht verwenden. Denn was passiert mit Linien die vertikal verlaufen ? Deren Steigung ist nicht definiert, sondern wird als unendlich angenommen ? Wir können mit diesem Wert aber keine Linie nachzeichnen (siehe Abbildung 2).
Neben der allgemeinen Formel der Gleichung gibt es in der Mathematik eine weitere Möglichkeit eine Gerade zu definieren: Mit der Hessechen Normalform.

$$p = x * \cos(phi) + y * \sin(phi)$$

p gibt dabei den Abstand der Linie zum Nullpunkt an.

Wenn wir diese Formel benutzen, können wir das Problem der vertikalen Linien lösen.

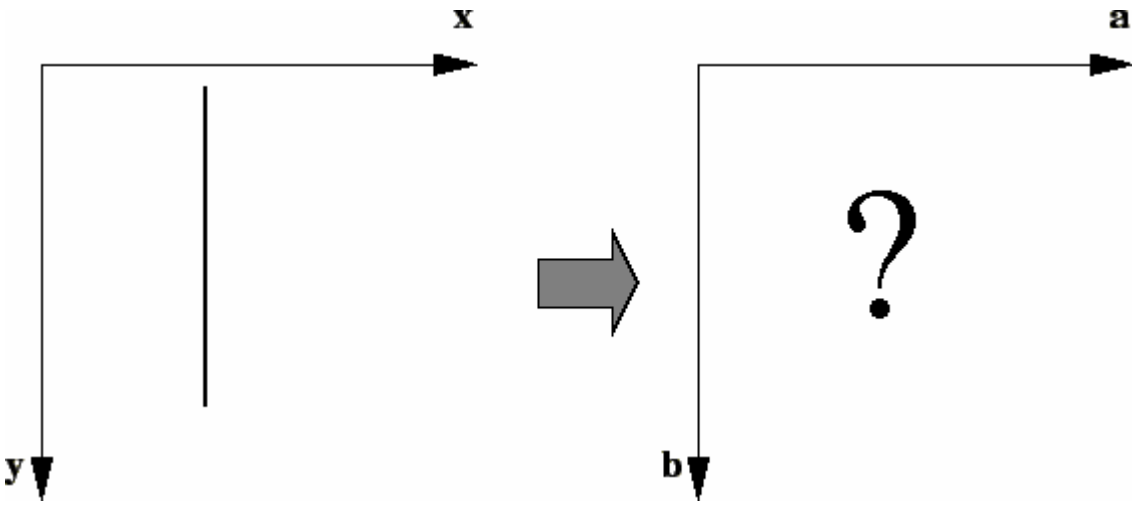


Abbildung 2 Welche Steigung hat eine vertikale Linie ?

Werden der Funktion die bekannten X und Y Koordinaten des Pixel mitgegeben sowie Winkel von –90° bis +90° Grad vorgegeben, kann der Abstand der Linie zum Nullpunkt berechnet werden.

Computermodell

Die bisherigen Ergebnisse werden nun auf ein Computermodell übertragen:

Würden wir unsere bisherigen Forschungen einsetzten, es entstünde eine Transformation jedes einzelnen Punktes in den Hough Raum (so wird der neue Raum genannt). Dazu würden wir den Abstand eines jeden Punktes ausrechnen und in ein neues Koordinatensystem einsetzen, das als neue X Achse das 2fache des maximalen Abstand zum Nullpunkt hätte und als Y Achse die Winkel von –90° bis +90°. Da mit Hilfe der Hesseche Normalform der Abstand einer Linie zum Nullpunkt nicht nur positiv, sondern je nach Quadrant des Koordinatensystems auch negativ sein kann, muss die x Achse 2 x die Länge des längsten Abstands sein.

Wie groß ist der maximalste Abstand ?
In Abbildung 3 ist der maximalste Abstand grün gezeichnet und *d'* benannt. Der maximalste Wert den *d'* annehmen kann ist nach dem Satz des Pythagoras ($a^2+b^2=c^2$) :

$$d' = ((\text{Höhe des Bildes})^2 + (\text{Breite des Bildes})^2)^{1/2}$$

Den Grund warum wir als Winkel –90 bis 90 annehmen ist der Tangens denn:

Die Steigung einer Geraden ist der Tangens des Winkels a zwischen Gerade und x-Achse mit $-90° \leq \alpha \leq 90°$.

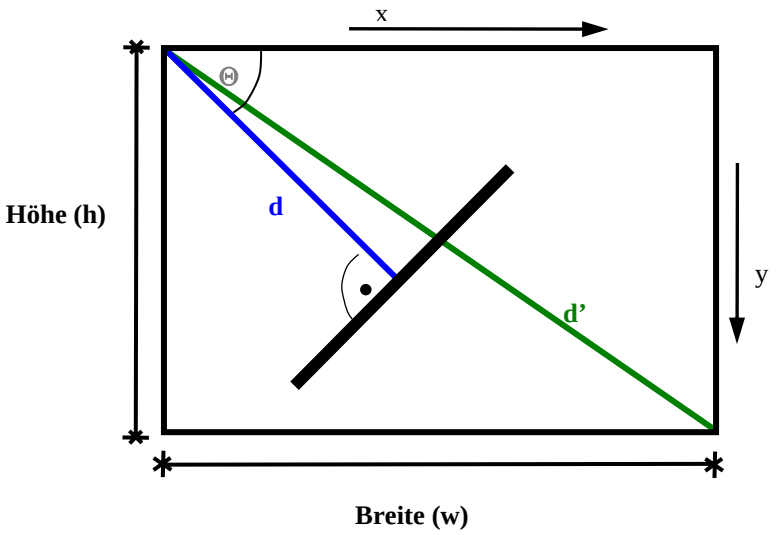


Abbildung 3 Eine Linie in einem Bild

Der Winkel Teta (grau) ist der Winkel zwischen der Linie zum Ursprung und der x –Achse. Die blaue Linie ist der Abstand eines Punktes der Linie im Bild. Die grüne Linie ist die Linie die maximal entstehen kann (also der größt möglichste Abstand).

Nachdem alle Punkte in den Hough Raum übertragen wurden (Abbildung 4), würden sich die so entstandenen Sinoiden irgendwo schneiden, wir würden den Schnittpunkt ablesen und hätten den Winkel sowie den dazugehörigen Abstand einer Linie zur x Achse bzw zum Ursprung.

Auf den Computer übertragen heißt das: Wir benötigen einen Array der Größe

$$180 \times 2 \times ((\text{Höhe des Bildes})^2 + (\text{Breite des Bildes})^2)^{1/2}$$

(ein Analyse des Speichervolumens sowie eine Laufzeitanalyse werden weiter unten besprechen). Des Weiteren brauchen wir drei for-Schleifen:

eine für die Abtastung der X Koordinate des Bildes,
eine für die Abtastung der Y Koordinate des Bildes,
sowie eine for Schleife für die Winkel von –90° bis 90°.

Am Anfang werden die Array-Elemente alle auf 0 gesetzt. Wird durch die Abtastung ein Bildpunkt ermittelt, wird für jeden Winkel mit Hilfe der Hess'chen Normalform der Abstand um Nullpunkt errechnet und die dazugehörige Arraystelle (Winkel und Abstand) inkrementiert.

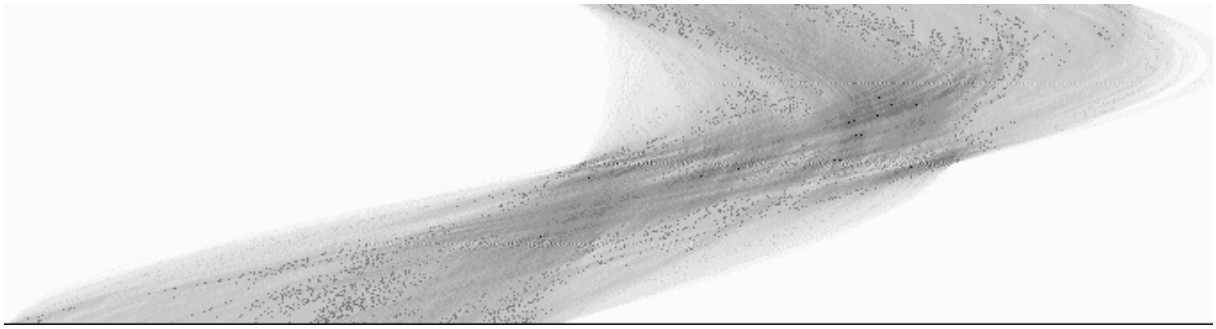
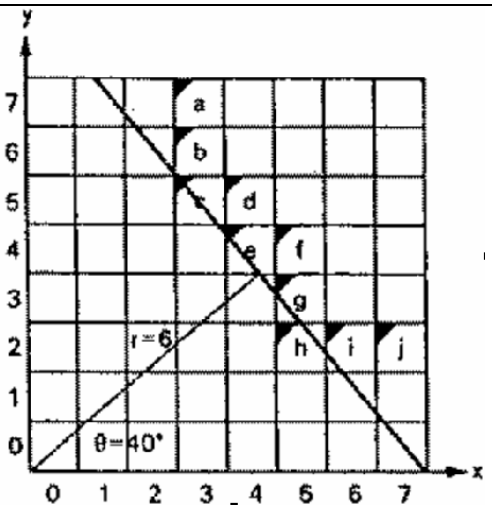


Abbildung 4 Hough-Roum mit Sinoiden

Die Werte an jeder Arraystelle werden Peaks genant. Wenn ein Peak einen bestimmten Schwellwert überschritten hat, wird mit Hilfe der Arraystelle die Steigung und ein Punkt der Linie berechnet.
Dazu berechnen wir den Ortsvektor eines Punktes der Linie und zeichnen mit Richtungsvektoren die Linie. Allerdings kann mit dieser Methode nicht der Start und Endpunkt der Linie bestimmt werden. Was allerdings auch Vorteile haben kann: Eine Linie, die unterbrochen ist, kann mittels der Hough - Transformation als eine Linie erkannt werden.

Ein Rechenbeispiel:



d \ θ	0°	20°	40°	60°	80°	100°	120°	140°	160°	180°
-10										
-9										
-8										
-7										j
-6									j	l
-5									i	fgh
-4								j	gh	de
-3								hi	f	abc
-2							j	g	de	
-1							hi	f	bc	
0							g	de	a	
1						hij	ef	c		
2						g	d	b		
3	abc				hij	ef	c	a		
4	de			h	g	cd	b			
5	fgh	abc deh	h	eij	cef	b	a			
6	i	fgi	bcd efgi	cd fg	bd	a				
7	j	j	ja	b	a					
8				a						
9										
10										

Wir nehmen an, dass wir ein Peak gefunden haben, der über dem Schwellwert liegt und im Hough Raum bei 60° und bei einem Abstand von 5 Pixeln zum Nullpunkt liegt.
Als erstes berechnen wir den Ortsvektor:

$OVx = \cos(\theta * PI/180) * d$	$OVx = 5$
$OVy = \sin(\theta * PI/180) * d$	$OVy = 0$

Dann drehen wir unseren Winkel θ um 90° damit wir genau auf die Linie kommen (die Hess'che Normalform hat den Vorteil, dass die Linie [der Abstand also] zum Nullpunkt senkrecht auf der von uns gefundenen Linie liegt). Da wir keine Start bzw. Endpunkte unserer Linie mittels der Hess'chen Normalform ausrechen können, nehmen wir als Faktor für die Richtungsvektoren einfach den maximalsten Abstand, den eine Linie zum Nullpunkt haben kann.

$$Startpunktx = OVx + \cos((\theta - 90) * PI/180) * -(MAX\_D * 2)$$
$$Startpunkty = OVy + \sin((\theta - 90) * PI/180) * -(MAX\_D * 2)$$
$$Endpunktx = OVx + \cos((\theta - 90) * PI/180) * (MAX\_D * 2)$$
$$Endpunkty = OVy + \sin((\theta - 90) * PI/180) * (MAX\_D * 2)$$

Schwellwert

Die Berechnung der Linien basiert darauf, dass zu erst der Hough-Raum berechnet wird. Die dort gefunden Ausschläge (Peaks) führen zu einer Möglichen Geraden. Der Schwellwert bestimmt nun, wie breit das Spektrum der Peaks ist. Wird ein niedriger Schwellwert benutzt, vergrößert sich das Spektrum der Peaks, und somit auch die Anzahl der gefundenen Kanten. Gute Ergebnisse liefert ein Schwellwert zwischen 0,5 und 0,8. Geht der Schwellwert unter diese Werte, werden noch mehr Kanten gefunden, die einen geringeren Peakausschlag haben, und die Anzeige füllt sich schnell mit den Markierungen und das Resultat wird unübersichtlich.

Dies erforderte eine zusätzliche Implementierung, um den Schwellwert während der Laufzeit verändern zu können.

Abbildung 5 Bild mit einem Schwellwert von 0.9

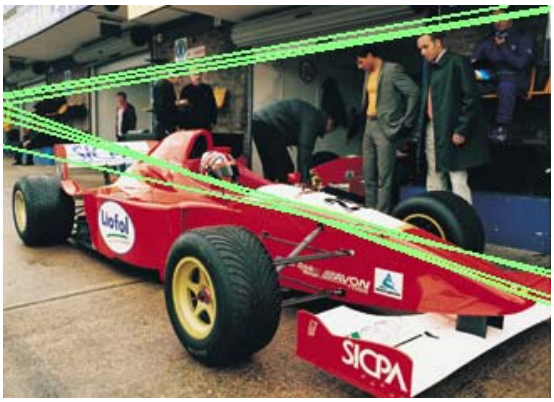


Abbildung 6 Bild mit einem Schwellwert von 0.83

Kantenmarkierung

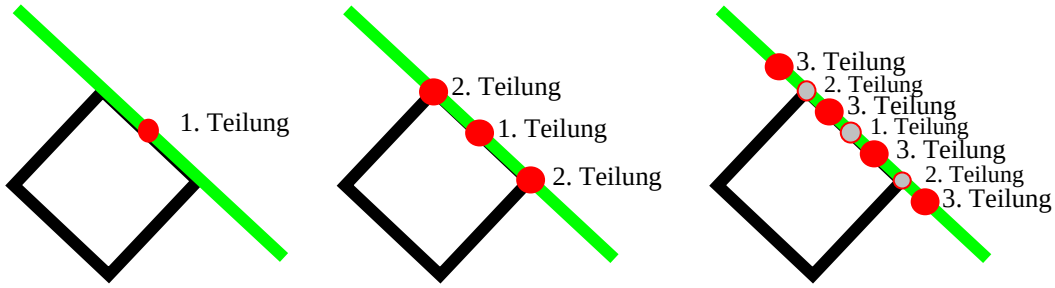
Die gefunden Kanten werden in unserer Anwendung mit einer durchgehenden grünen Linie markiert. Leider lässt sich so die genaue Position der Kante jedoch nicht festlegen, die Markierung zeigt nur an, dass sich auf dieser Linie die gefundene Kante befindet. Der Algorithmus kennt die genaue Start – und End – Position der Kanten in dem Bild leider nicht. Die Länge der Kanten dagegen ist die Höhe der Peaks, und somit bekannt.
Nach Recherchen haben wir ein Verfahren gefunden, in dem ein „Schiffchen“ die Markierung entlang läuft, und prüft, ob es sich noch auf der gefunden Kante befindet, indem die Pixel verglichen werden, oder nicht. Dieses Verfahren ist relativ langsam.

Ein anderes Verfahren haben wir uns bei der Implementierung überlegt. Von diesem erwarten wir eine bessere Laufzeit.

Die Idee sieht wie folgt aus:

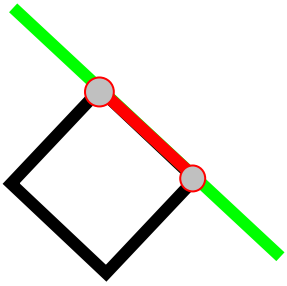
Die gefundene Kante befindet sich irgendwo auf der Markierung. Daraus folgt, wenn man die Markierung absucht, wird man auch die Kante finden. Die Suche soll bis zu einem bestimmten Grad rekursiv ablaufen.

Unsere Idee sieht eine Teilung der Markierung vor. Das heißt, dass die grüne Linie im Bild rekursiv geteilt wird.



Nach jeder Teilung soll auf dem Teilungspunkt überprüft werden, ob dieser sich noch auf der Kante befindet, und ob die Pixel beidseits auch noch zur Linie gehören.

- Liegt der Punkt auf der Kante, und sind beidseits noch Pixel der Kante, so wird wieder geteilt
- Sollte eine vom Teilungspunkt weggehende Seite nicht mehr auf der Kante liegen, so ist der Teilungspunkt das letzte Pixel der Kante. In diesem Falle müsste man von dem Teilungspunkt einfach nur noch die Länge der Kante auf der anderen Seite des Punktes markieren. Damit wäre die Suche beendet



Ein Problem für diesen Algorithmus werden geteilte Linien. So müsste man dann die einzelnen Teilungspunkte überprüfen , und dann auch jeden einzelnen auswerten. Das ganze Verfahren müsste dann bis zu einem bestimmten Grad geführt werden, bis man sicher sich sein kann, dass die Kanten nun alle eindeutig gefunden wurden. Ab einem bestimmten Teilungs-Grad wäre auch die Anwendung des Schiffchens denkbar. Da dieses dann nicht mehr die gesamte Kante absuchen muß, sondern nur einen kleinen Teil.

Die Vorbereitung der Bilder

Bevor wir die Hough Transformation anwenden können, müssen die Bilder vorbereitet werden. Das heißt in unserem Fall, die Umwandlung der Bilder in Grauwert – Bilder und die Anwendung eines Filters. Eine Bilddatei hat normalerweise die drei Grundfarben Rot, Grün und Blau, sowie einen Alphawert, der die Helligkeit des Bildes bestimmt. Um das Bild filtern zu können wird es vorher in ein Graubild überführt. Durch das filtern des Bildes (wir nehmen hier an, wir würden das Bild mittels canny Filter filtern) entsteht ein „binär Bild“, das nur zwei Zustände kennt. Eigentlich werden die Farbwerte der drei Grundfarben entweder auf 0 oder 255 gesetzt (im folgenden gehen wir aber der Einfachheit halber von einem Binärbild aus, daß entweder nur 0 oder 1 kennt). Wird beim Abtasten des gefilterten Binärbildes eine 1 gelesen, so ist im original Bild an der selben Stelle ein Übergang von einem dunklen Teil des Bildes in einen hellen Teil des Bildes. Berechnet werden solche Übergänge durch Ableitungen die auf Basis der Gaußfunktion beruhen [2]. Im genaueren bedeutet das eine Ableitung des Binomial-Glättungsfilters. Dies ist ein Kapitel für sich und soll hier nicht weiter betrachtet werden. Näheres sie unter [2].

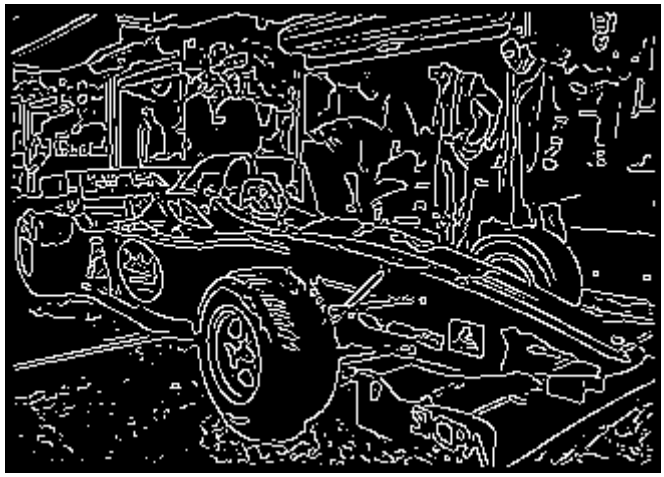


Abbildung 7 gefiltertes Bild, Canny - Filter

Kantenmarkierung

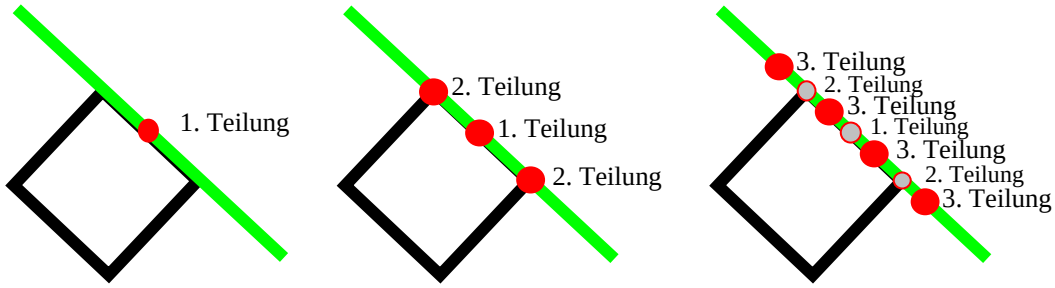
Die gefunden Kanten werden in unserer Anwendung mit einer durchgehenden grünen Linie markiert. Leider lässt sich so die genaue Position der Kante jedoch nicht festlegen, die Markierung zeigt nur an, dass sich auf dieser Linie die gefundene Kante befindet. Der Algorithmus kennt die genaue Start – und End – Position der Kanten in dem Bild leider nicht. Die Länge der Kanten dagegen ist die Höhe der Peaks, und somit bekannt.
Nach Recherchen haben wir ein Verfahren gefunden, in dem ein „Schiffchen“ die Markierung entlang läuft, und prüft, ob es sich noch auf der gefunden Kante befindet, indem die Pixel verglichen werden, oder nicht. Dieses Verfahren ist relativ langsam.

Ein anderes Verfahren haben wir uns bei der Implementierung überlegt. Von diesem erwarten wir eine bessere Laufzeit.

Die Idee sieht wie folgt aus:

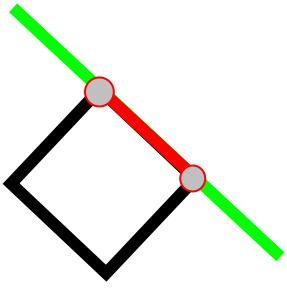
Die gefundene Kante befindet sich irgendwo auf der Markierung. Daraus folgt, wenn man die Markierung absucht, wird man auch die Kante finden. Die Suche soll bis zu einem bestimmten Grad rekursiv ablaufen.

Unsere Idee sieht eine Teilung der Markierung vor. Das heißt, dass die grüne Linie im Bild rekursiv geteilt wird.



Nach jeder Teilung soll auf dem Teilungspunkt überprüft werden, ob dieser sich noch auf der Kante befindet, und ob die Pixel beidseits auch noch zur Linie gehören.

- Liegt der Punkt auf der Kante, und sind beidseits noch Pixel der Kante, so wird wieder geteilt
- Sollte eine vom Teilungspunkt weggehende Seite nicht mehr auf der Kante liegen, so ist der Teilungspunkt das letzte Pixel der Kante. In diesem Falle müsste man von dem Teilungspunkt einfach nur noch die Länge der Kante auf der anderen Seite des Punktes markieren. Damit wäre die Suche beendet



Ein Problem für diesen Algorithmus werden geteilte Linien. So müsste man dann die einzelnen Teilungspunkte überprüfen , und dann auch jeden einzelnen auswerten. Das ganze Verfahren müsste dann bis zu einem bestimmten Grad geführt werden, bis man sicher sich sein kann, dass die Kanten nun alle eindeutig gefunden wurden. Ab einem bestimmten Teilungs-Grad wäre auch die Anwendung des Schiffchens denkbar. Da dieses dann nicht mehr die gesamte Kante absuchen muß, sondern nur einen kleinen Teil.

Leider lässt sich nicht vorhersagen, an welcher Stelle im Hough Raum die Peaks akkumuliert werden müssen. Aber vielleicht gibt es eine andere Möglichkeit den Hough Raum nur auf die Kurven zu beschränken. Wir haben leider keine andere Möglichkeit gefunden.

Eine weiterer interessanter Aspekt im Zusammenhang mit dem Speicherplatzverbrauch bei der Hough Transformation, ist der Zusammenhang zwischen Bildgröße und Hough Raum Größe. In Abbildung 10 sehen wir einen durch Excel erstellten Graf, der uns zeigt das die Größe des Hough Raums ab einer bestimmten Bildgröße überholt wird. Dieser Graf wurde auch unter Annahme von quadratischen Bildern erstellt.

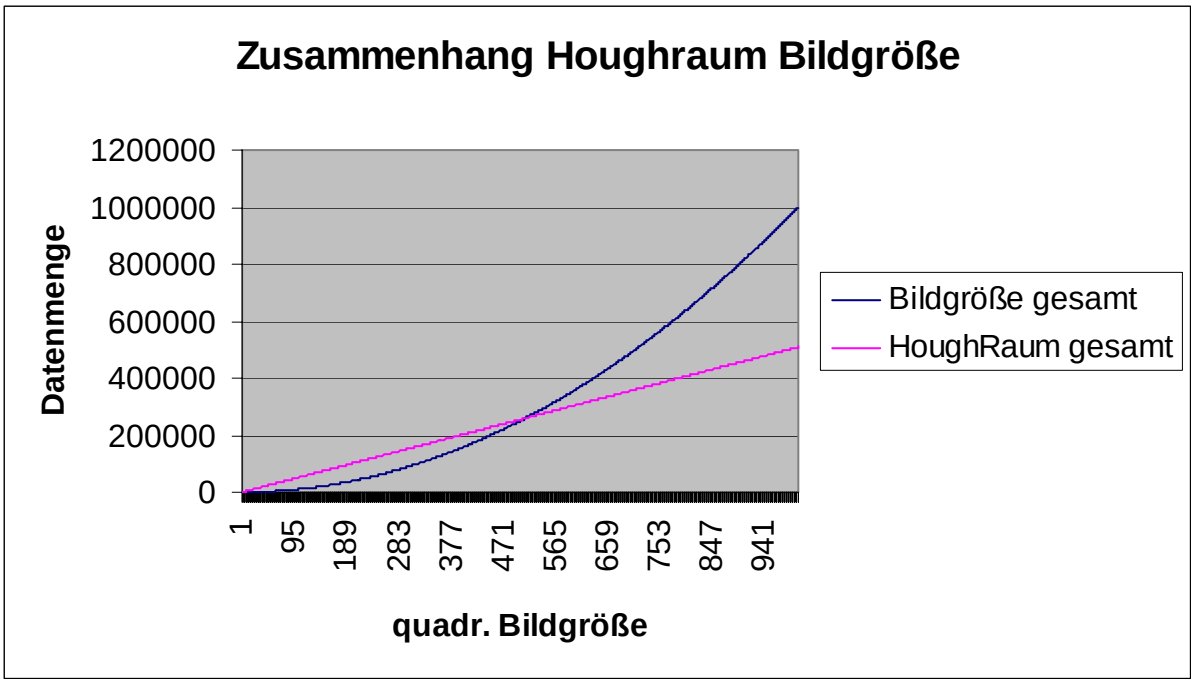


Abbildung 10

Ab einer ungefähren Bildgröße von 510 x 510 Bildpunkten, hat die Kurve "Bildgröße gesamt", die Kurve "Hough Raum gesamt" überholt. Das heißt in einer Fertigungsstrasse, die Bilder analysiert, die unter einer Bildgröße von 510 x 510 Bildpunkten liegen, ist der Hough Raum der Speicherintensivste Array.

In unserer Anwendung kommt eine weitere Datenstruktur zum Einsatz. Die zurück berechneten Linien werden in einem Stack gehalten und später zum zeichnen wieder ausgelesen. Da wir nicht vorhersehen können, wie viele Linien gefunden werden (das ist Abhängig vom Schwellwert), können wir auch keine Angaben zur Speichergröße machen.

2. Gibt es Verbesserungsvorschläge ?

Auch nach längerer Recherche konnten wir keine Alternative zum Hough Raum finden. Es existieren demnach keine Verbesserungsvorschläge.

Laufzeitanalyse

Zu unserem Erstaunen stellten wir bei der Laufzeitanalyse fest, entgegen unserer Vermutung [am Anfang war sie $O(n^3)$], das die tatsächliche Laufzeit unseres Algorithmus O(n) ist. Nach Auswertung unserer Testdaten, wofür wir eigens eine Testmethode entwickelt haben (PowertestWerte), konnten wir auch eine Erklärung für diese Laufzeit finden.
Die drei FOR Schleifen in unserer BerechneHoughRaum Klasse könnten wir in zwei FOR Schleifen packen. Denn die Abtastung des Bildes, derzeit mit jeweils einer For Schleife für die Zeilen und eine für die Spalten, könnten in einer einzigen FOR Schleife gepackt werden, die über die Bildbreite hinaus zählt und damit automatisch in die nächste Zeile gelangt. Das Bild wird nämlich auch in nur einem Ein-Dimensionalen Array gehalten. Die Winkel, die für die Berechnung der Hess'chen Normalform benötigt werden, sind dann nur noch eine multiplikative Konstante, die bei der O-Notation aber weggelassen werden kann.

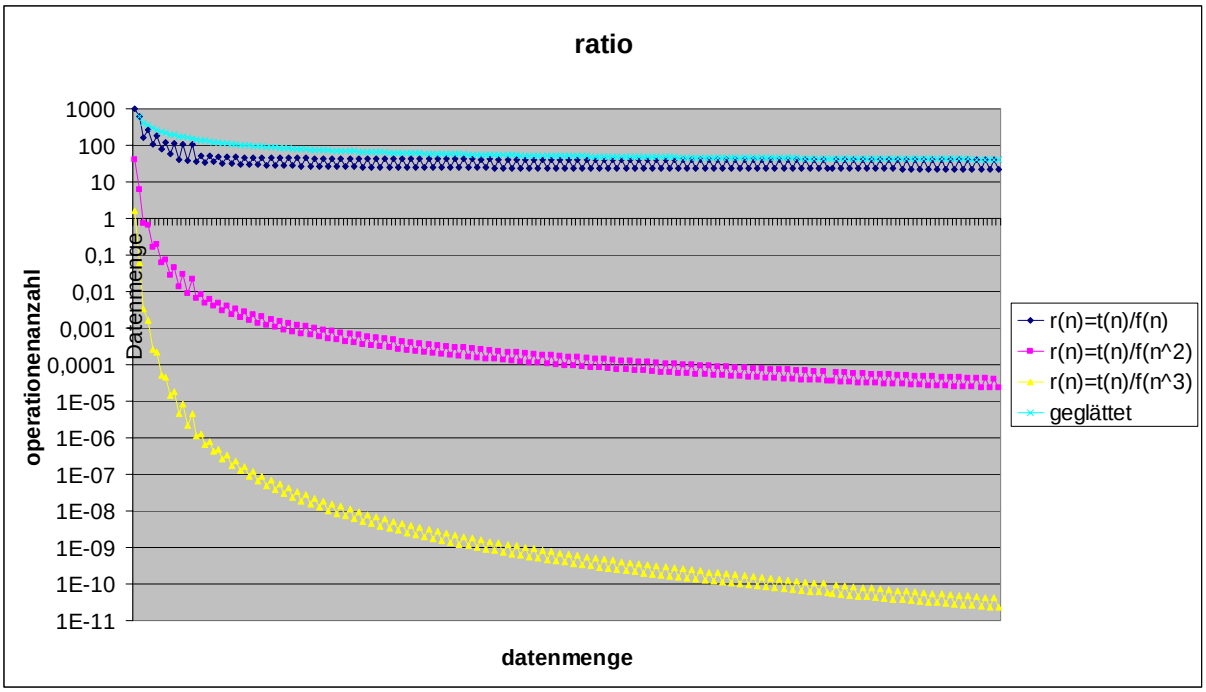


Abbildung 11 Ratio Test

Klassen und Interfaces

HoughTransformationGUI.java:

Ist die Hauptklasse, d.h. diese Klasse enthält unsere main Methode. Außerdem implementiert diese Klasse unsere GUI, mit Menüsteuerung, Fenster und vielen GUI spezifischen Elementen, sowie ein Teil der TestMethoden (Ratio, Power, etc…)

IBildBearbeitung.java:

Das ***I*** vor dem Namen ist ein Indiz für Interface. Diese Schnittstelle ist erstellt worden, damit die GUI unsere berechneten verschiedenen Bilder von den Klassen, die diese Schnittstelle einbinden, zurück erhalten kann.

BerechnungBildFilter.java:

Mit Hilfe dieser Klasse, wird das gefilterte Bild (durch Aufruf einer Methode der Klasse EdgeDector) an unsere GUI weitergeleitet. Dadurch erreichen wir eine Trennung da wir die Methode der Klasse EdgeDetector nicht direkt aufrufen wollen.

EdgeDetector.java:

Dies ist eine Klasse, die nicht von uns implementiert wurde. Sie erstellt ein gefiltertes Bild, wo die Kantenübergänge hervorgehoben sind.

EdgeDetectorExceptionsjava:

Enthält alle Fehlermeldungen, die bei der Berechnung des gefilterten Bildes auftreten können.

BerechnungHoughRaum.java:

In dieser Klasse wird der Hough Raum berechnet, der später zur Auswertung der Linien benötigt wird.

BerechnungHough.java:

Der von der Klasse BerechnungHoughRaum berechnete Hough Raum wird hier wieder in Linien zurückgerechnet, wenn die Peaks einen bestimmten Schwellwert überschreiten.

MyImageObserver.java:

Diese Klasse enthält eine einzige Methode, die von unserer GUI automatisch aufgerufen wird um nachzufragen ob sich die Bildgröße geändert hat. Da wir keine Veränderung des Bildes vornehmen, ist die Antwort immer true. Damit wird erst die Darstellung des Bildes ermöglicht.

Powertest.java:

Unsere eigene Testklasse. Die main Methode generiert ein Bild und ruft die Methoden zur Hough Transformation auf. Außerdem wurde die Klasse dazu genutzt um den Speicherplatzverbrauch zu testen.

Sonstiges:

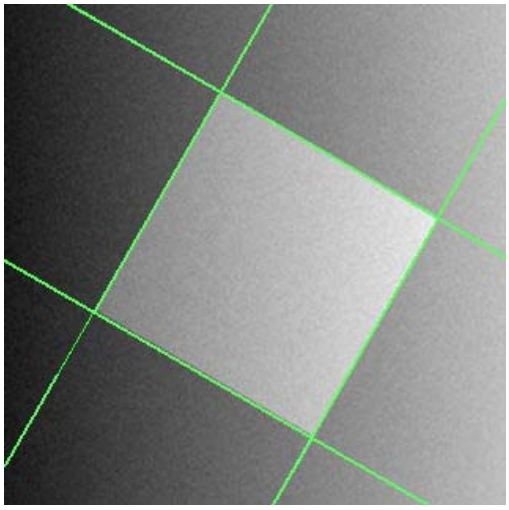
Neben dieser Dokumentation befinden sich weitere Dateien in unserem Verzeichnis. Zum einen sind dies Excel Tabellen, die unsere Testergebnisse dokumentieren, Beispielbilder, sowie die Anwendung.

Quellen

[1] Patentschrift Hough: **US-Patent No. 306965418**

[2] **Bernd Jähne, Digitale Bildverarbeitung,**
6., überarbeitete und erweiterte Auflage, Springer Verlag,
ISBN 3-540-24999-0 © Springer-Verlag Berlin Heidelberg 2005

[3] **Guido Krüger, Handbuch der Java-Programmierung**,
3. Auflage, HTML-Ausgabe 3.0 ·
ISBN 3-8273-1949-8 © 1998-2002 Guido Krüger Addison-Wesley, 2002,



Learning by Doing - Filter

Das zu bearbeitende Bild kann nur in schwarz –weiß verarbeitet werden, daher wurde ein Filter ausgesucht, der das Bild s/w macht, und dazu noch die Kanten hervorhebt. Der zu erst von uns verwendete Filter (Sobel) funktionierte gut, hatte aber schwerwiegende Nachteile. Dadurch, dass der Filter die Kanten zu sehr betonte und hervorhob, führte dies zu einer fehlerhaften späteren Berechnung der Linien.

Aus diesem Grund entschlossen wir uns den Filter durch einen anderen zu ersetzten. Der neu eingesetzte Filter ist der Canny-Filter. Dieser Filter führt die Berechnung auf eine andere Art und Weise, und erreicht dadurch ein besseres Ergebnis. Mit diesem Ergebnis sind Linien nicht mehr von der unmittelbaren Nachbarschaft anderer Pixel abhängig. Als Endresultat funktioniert die Erkennung der Kanten durch die Hough – Transformation besser.

Learning by Doing – Linien zeichnen

Ein weiteres Problem war die Implementierung einer Methode die Linien zeichnet. Vor allen Dingen haben die mathematischen Berechnungen und die dazu gehörigen Fallunterscheidungen einiges Kopfzerbrechen bereitet.

Learning by Doing - Schwellwert

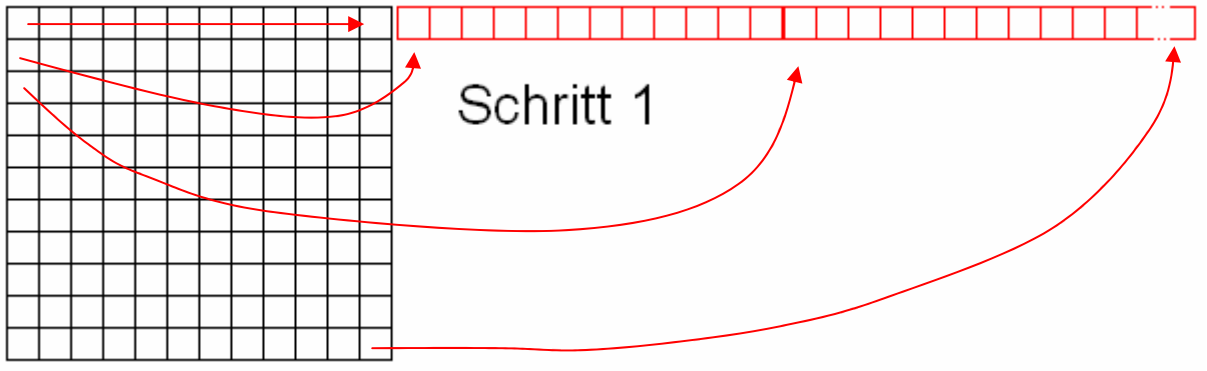
Die Berechnung der Linien basiert darauf, dass zu erst der Hough-Raum berechnet wird. Die dort gefunden Ausschläge (Peaks) führen zu einer Möglichen Geraden. Der Schwellwert bestimmt nun, wie breit das Spektrum der Peaks ist. Wird ein niedriger Schwellwert benutzt, vergrößert sich das Spektrum der Peaks, und somit auf die Anzahl der Linien.

Dies erforderte eine zusätzliche Implementierung, um den Schwellwert während der Laufzeit verändern zu können.

Learning by Doing - Laufzeit

Erste Annahme war, das die Laufzeit n^3 beträgt.
Nach der Implementierung, und der Durchführung des Power bzw. Ratio Tests, und die spätere Auswertung der Ergebnisse kamen wir zum Schluss, dass unsere Annahme von einer Laufzeit von n^3 zu hoch angesetzt war. Der Algorithmus hat eine Laufzeit von O(n).

Dieses Phänomen erklärt sich dadurch, dass die Pixel der gefilterten Bilder als ein Stream hintereinander verarbeitet werden. Die spätere Berechnung des Abstands mit von uns vorgegebenen Winkeln (0° - 180°) auf das jeweilige Pixel ist nur eine Aufaddierung, und verändert die Laufzeit nicht wirklich.


Schritt 1