Technical Report

Pattern Recognition and Image Processing Group
Institute for Computer Aided Automation
Vienna University of Technology
Treitlstr. 3/1832
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801-18351
Fax: +43 (1) 58801-18392
E-mail: e9327010@student.tuwien.ac.at

PRIP-TR-59 19. Mai 1999

Lineare Hough-Transformation und Drehtellerkalibrierung

Srdan Tosovic

Abstract

In dieser Arbeit wird ein Verfahren zur Kalibrierung eines Aufnahmesystems vorgestellt, das als Ergebnis die Transformationsmatrix liefert, die die Bildkoordinaten des Aufnahmeobjekts in die Objektkoordinaten umwandelt. Dabei besteht das Aufnahmesystem aus einem Drehteller und einer Kamera, deren optische Achse in der Drehtellerebene liegt. Da das Verfahren sehr stark auf der linearen Hough-Transformation beruht, wird die lineare Hough-Transformation im ersten Teil dieser Arbeit beschrieben. Danach folgt die Beschreibung des Kalibrierungsverfahrens.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Das in dieser Arbeit vorgestellte Verfahren zur Kalibrierung eines aus einer Kamera und einem Drehteller bestehenden Aufnahmesystems (im weiteren als Drehtellerkalibrierung bezeichnet) wurde als Vorbereitungsschritt für einen anderen Task, "*Shape from Silhouette*", entwickelt. Bei dem Task soll ein Objekt auf einen Drehteller gestellt und mit einer Kamera von verschiedenen Blickwinkeln aufgenommen und aufgrund dieser Aufnahmen ein 3D-Model des Objekts erfaßt werden. Damit dies möglich wird, muß man das Aufnahmesystem kalibrieren, d.h., innere und äußere Kamaraparameter [Tsa86] bezüglich des Objektkoordinatensystems bestimmen. Wenn ein Drehteller ein Teil des Aufnahmesystems ist, ist die Bestimmung seiner Rotationsachse eine wichtige Aufgabe des Kalibrierungstasks, da sich das Objektkoordinatensystem vor der Kamera dreht. Statt bei jeder Aufnahme das Aufnahmesystem neu zu kalibrieren, ist es effizienter die Rotationsparameter zu berücksichtigen und mit deren Hilfe den neuen Zustand des Systems zu beschreiben.

Der entwickelte Kalibrierungstask nimmt an, daß die optische Achse [Har91] orthogonal zur Rotationsachse des Drehtellers ist und (nahezu) auf der Drehtellerebene liegt. Warum? Damit man im darauffolgenden Task, "*Shape from Silhouette*", durch die Rotation eines Objekts auf dem Drehteller möglichst viel Information über das Objekt erhält – wäre z.B. die optische Achse der Kamera orthogonal zur Drehtellerebene, dann würde die Rotation des Drehtellers überhaupt keine neuen Informationen über das auf ihm stehende Objekt liefern – das Objekt wäre immer nur von einer Seite her aufgenommen.

In den nächsten Abschnitten wird der Kalibriervorgang detailliert beschrieben. Im Abschnitt 2 wird das Aufnahmesystem näher beschrieben, Abschnitt 3 befaßt sich mit der Hough-Transformation, die einen wesentlichen Baustein des Tasks darstellt. Im Abschnitt 4 wird auf das Kalibrierungsverfahren eingegangen und im Abschnitt 5 werden die durchgeführten Tests beschrieben und die Ergebnisse analysiert. Abschließend wird im Abschnitt 6 eine Zusammenfassung mit Ausblicken zur Verbesserung des Verfahrens gegeben.

Im Anhang A wird auf die Implementierungsdetails der entwickelten Tasks eingegangen, und Anhang B beschreibt die Benutzeroberflächen der Tasks und erklärt, wie sie zu verwenden sind.

2. Das Aufnahmesystem

In diesem Abschnitt werden zunächst die Anforderungen an das Aufnahmesystem angegeben, damit das entwickelte Kalibrierungsverfahren verwendet werden kann, danach folgt die Beschreibung des verwendeten Aufnahmesystems.

2.1 Anforderungen

An Geräten braucht man folgendes:
- eine schwarz/weiß CCD-Kamera,
- einen Drehteller, dessen relative Drehwinkel auf 1° bis 2° genau spezifiziert werden kann.

Als Kalibrierobjekt braucht man:
- einen regelmäßigen Kegel bekannter Höhe, mit dem seitlichen Winkel 30° bis 60° (am besten 45°, damit die Schenkel des Kegels orthogonal zueinander sind und somit die Berechnung der Koordinaten der Kegelspitze numerisch stabil wird).

Zusätzlich muß auch folgendes gelten: die optische Achse der Kamera muß (nahezu) auf der Drehtellerebene liegen, sodaß die untere Seite des Kegels in einem von der Kamera aufgenommenen Bild als eine gerade Linie erkennbar ist. Dies ist notwendig, weil der Kalibriervorgang die x-Achse des Objektkoordinatensystems durch diese Linie (untere Seite des Kegels) approximiert .

Um die Genauigkeit des Kalibriervorgangs zu erhöhen, sollte größtmöglicher Kontrast zwischen dem aufzunehmenden Objekt und dem Hintergrund hergestellt werden, bzw. das Objekt sollte möglichst dunkel und der Hintergrund möglichst hell erscheinen (oder umgekehrt), was am besten durch die Beleuchtung des Objekts von hinten (von der Kamera aus gesehen) zu erzielen ist.

2.2 Das verwendete System

Das verwendete Aufnahmesystem besteht aus folgenden Geräten:
- einer schwarz/weiß CCD-Kamera, mit der Auflösung 768x572 Pixel und der fokalen Länge 16 mm,
- einem Drehteller, mit dem Radius R=250 mm.

Als Kalibrierobjekt wird ein Kegel der Höhe H=80 mm und des seitlichen Winkels α=45° verwendet.

Die geometrische Anordnung der Kamera und des Drehtellers ist in der Abbildung 2.1 gezeigt.

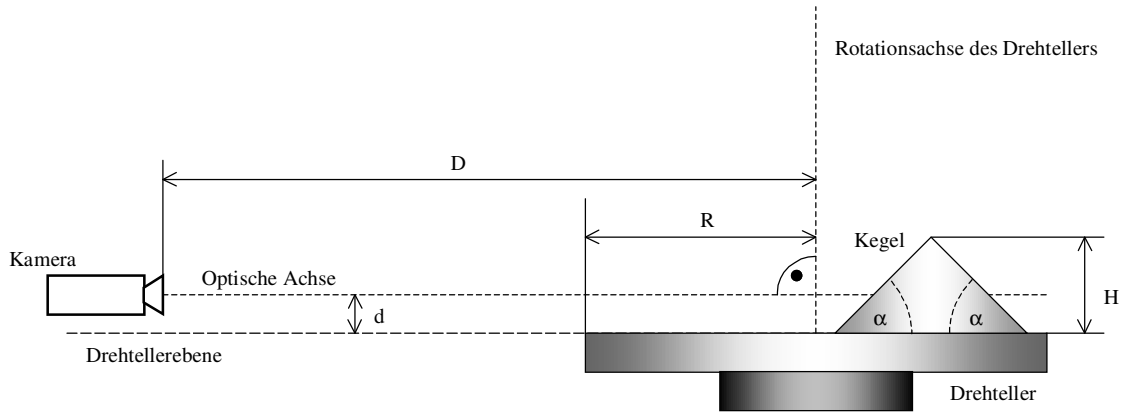


Abbildung 2.1: das Aufnahmesystem

Wie die Abbildung 2.1 zeigt, liegt die optische Achse der Kamera knapp oberhalb der Drehtellerebene und orthogonal zur Rotationsachse des Drehtellers, mit dem Abstand *d*. Idealerweise sollte dieser Abstand 0 sein, da in dem Fall die untere Seite des Kegels auf eine Gerade im Bild abgebildet wird. Im verwendeten System ist er ca. 10 mm. Die Entfernung *D* zwischen dem optischen Zentrum der Kamera und der Rotationsachse des Drehtellers ist ca. 130 cm (wie viel genau, wird vom Kalibrierungstask ermittelt).

3. Lineare Hough-Transformation

Dieser Abschnitt befaßt sich mit der linearen Hough-Transformation [Dud72], da sie einen wesentlichen Teil des implementierten Kalibrierungsverfahrens darstellt und auch als eigener Task implementiert wurde. Im ersten Teil des Abschnittes wird ein Überblick über sie gegeben und im zweiten Teil wird die konkrete Realisierung beschrieben. Im wesentlichen beruht dieser Abschnitt auf dem Buch "Bildverarbeitung Ad Oculos" von H. Bässmann und Ph. W. Besslich, Abschnitt "Hough-Transformation" [Bäs93].

3.1 Überblick

Lineare Hough-Transformation ist ein Spezialfall der allgemeinen Hough-Transformation [Hou62], die dafür verwendet wird, Objekte mit einem bestimmten Merkmal aus dem Eingabebild zu extrahieren und in analytischer Form darzustellen. Dabei kann es sich um regelmäßige geometrische Objekte handeln, wie Geraden, Kreise oder Ellipsen, aber auch um beliebige parametrisierbare Kurven.

Bei der linearen Hough-Transformation versucht man Geradenstücke im Eingabebild zu finden. Die Ausgabe sind die gefundenen Geraden im mathematischen Sinne, d.h., in ihrer analytischen Parameterdarstellung. Die intuitive Parameterdarstellung einer Gerade wäre, die Steigung A und der y-Achsenabschnitt B der Gerade anzugeben (siehe Abbildung 3.1a):

$$y = A\,x + B$$

Diese Darstellung hat aber den Nachteil, daß senkrechte Geraden nicht dargestellt werden können, und auch, daß die beiden Parameter für nahezu senkrechte Geraden unendlich groß werden können. Diese beiden Nachteile werden durch die Darstellung in der sog. Hesseschen Normalform behoben. Dort wird eine Gerade durch ihren senkrechten Abstand r zum Koordinatenursprung und den Winkel θ zwischen r und der x-Achse beschrieben (siehe Abbildung 3.1b):

$$r = x\cos\theta + y\sin\theta$$

Dabei liegt θ im Intervall [0, π). Wenn der Koordinatenursprung in die Mitte des Eingabebildes der Größe $w \times h$ gesetzt wird, liegt r im Intervall

$$[-\sqrt{\left(\frac{w}{2}\right)^2+\left(\frac{h}{2}\right)^2}, \sqrt{\left(\frac{w}{2}\right)^2+\left(\frac{h}{2}\right)^2}]$$

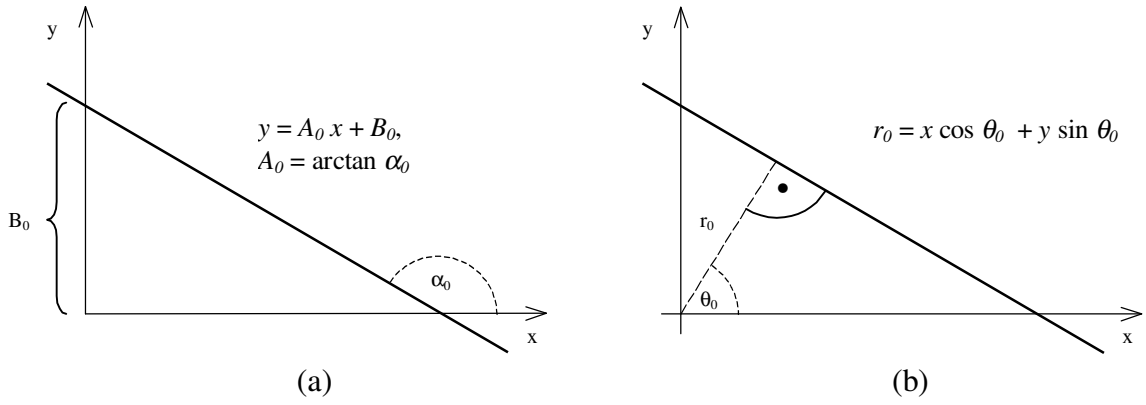


Abbildung 3.1: Explizite (a) und Hessesche (b) Darstellung einer Gerade

D.h., man hat einen begrenzten (r, θ)-Raum geschaffen, in dem jeder Punkt einer Gerade entspricht. Das bringt den Gedanken nahe, das Ergebnis der linearen Hough-Transformation als digitales Bild darzustellen, wo jedes Pixel mit dem Wert größer Null eine im Eingabebild detektierte Gerade darstellt, oder genauer gesagt, wo der Wert jedes Pixels gleich der Anzahl der im Eingabebild gefundenen Punkte ist, die auf der dem Pixel entsprechenden Gerade liegen. Dabei wird dieses Ausgabebild Akkumulator genannt, und seine Pixel Akkumulatorzellen. Abbildung 3.2 veranschaulicht die Darstellung einer Gerade im (r, θ)-Raum.

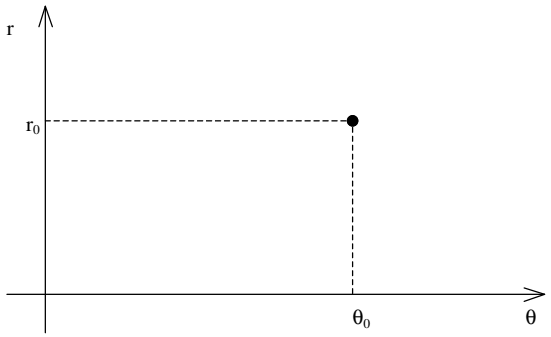


Abbildung 3.2: Darstellung einer Gerade im (r, θ)-Raum

Wie wird bestimmt, auf welcher Gerade ein Punkt des Eingabebildes liegt? Zuerst muß man auf das Eingabebild einen Gradientenoperator [Son93] anwenden, der sowohl Gradientenbetrag als auch Gradientenrichtung als Ergebnis liefert. Erst dann kommt die lineare Hough-Transformation zum Einsatz. Für jedes "helle" Pixel des Gradientenbetragsbildes, d.h., für jedes Pixel, das auf einer Kante liegt, wird im Gradientenrichtungsbild die Richtung der Kante für dieses Pixel abgelesen, und aus diesen Daten (Koordinaten des Pixels und die Richtung der Kante) werden die entsprechenden Parameter r und θ berechnet und die korrespondierende Akkumulatorzelle um 1 inkrementiert.

Wenn alle Kantenpunkte des Eingabebildes auf diese Art bearbeitet sind, ist die lineare Hough-Transformation abgeschlossen. Eventuell kann man noch eine "Säuberung" des Akkumulators durchführen, d.h., für jeden Cluster [Har91] von positiven Akkumulatorzellen, also für gefundene Geraden, die sich in r und θ nur ganz wenig unterscheiden, eine repräsentative Gerade finden (d.h., im Prinzip eine Art Schwerpunkt des Clusters finden), und den Wert der ihr entsprechenden Akkumulatorzelle hervorheben, was die nachfolgende aufgabenspezifische Analyse des Akkumulators erleichtert – es genügt lokale Maxima des Akkumulators zu finden, um die Parameter der im Eingabebild gefundenen Geraden abzulesen.

3.2 Realisierung

Die im Rahmen dieses Praktikums implementierte lineare Hough-Transformation wurde in der Programmiersprache *C* und mit der Hilfe des Bildverarbeitungsystems *Khoros* [Khoros] realisiert, unter Betriebssystem *Linux* [Linux].

Eingabe für das Programm ist das Eingabebild, in dem die Geraden zu finden sind, und andere Eingabeparameter, die mit sinnvollen Defaultwerten versehen sind. Dazu zählen die Kernels, die für den Gradientenoperator in x- und y-Richtung verwendet werden, der Schwellwert des Gradientenbetragsbildes, die Breite und der Schwellwert des Akkumulatorbildes und der Kernel, der für die Akkumulatorsäuberung verwendet wird. Für eine genaue Beschreibung aller Ein- und Ausgabeparameter siehe Anhang B, Abschnitt B.3.3.

Ausgabe sind der Akkumulator (als ein digitales Bild) und eine Textdatei mit Parametern der gefundenen Geraden.

Den Ablauf des Programms kann man auf folgende Phasen aufteilen:

- Bildung des Gradientenbetrags- und Gradientenrichtungsbildes durch die Anwendung eines Gradientenoperators auf das Eingabebild,
- lineare Hough-Transformation und Bildung des Akkumulators,
- Säuberung des Akkumulators durch die Anwendung eines Laplacian-of-Gaussian [Son93] (oder kurz LoG-) oder eines benutzerdefinierten Operators,
- optionale Ausgabe der Parameter der gefundenen Geraden in eine spezialformatierte Textdatei.

Auf die Implementierungsdetails wird im Anhang A näher eingegangen.

4. Drehtellerkalibrierung

In diesem Abschnitt wird das Kalibrierungsverfahren beschrieben. Er besteht aus drei Teilen:

- Bestimmung der Rotationsachse des Drehtellers,
- Bestimmung der Transformationsmatrix für die Umwandlung des Bildkoordinatensystems in das Objektkoordinatensystem und
- Berechnung der Entfernung zwischen dem optischen Zentrum der Kamera und der Rotationsachse des Drehtellers.

Es wird vom idealen Kameramodell ausgegangen, d.h., es wird angenommen, daß alle Eingabebilder unverzerrt sind.

4.1 Bestimmung der Rotationsachse

Das Kalibrierobjekt, der Kegel, wird auf den Drehteller gestellt, möglichst weit vom Rotationszentrum entfernt sodaß der Kegel gerade noch in jedem von der Kamera aufgenommenen Bild vollständig liegt (siehe Abbildung 4.1). Dann wird der Drehteller um gewünschte Winkelgrößen gedreht und der Kegel mit der Kamera aufgenommen. Die Winkelschrittweite bei der Drehung muß nicht konstant sein, aber es muß gelten, daß es zu jedem aus dem Winkel α aufgenommenen Bild ein Bild geben muß, das aus dem Winkel $\alpha+180°$ aufgenommen wurde. Wenn das nicht gilt, kann das Verfahren falsche Ergebnisse liefern.

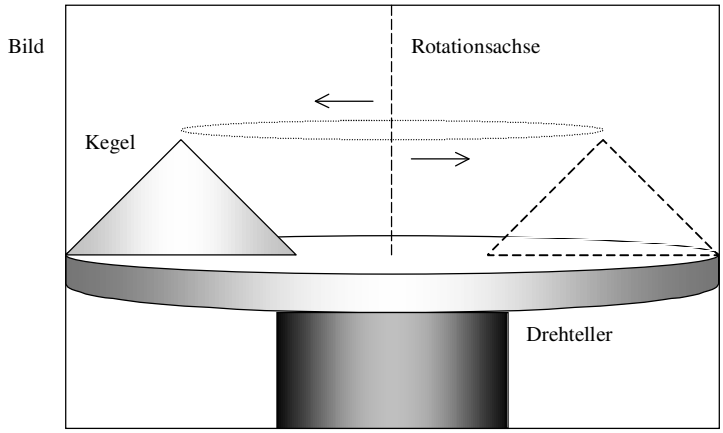


Abbildung 4.1: Ideale Stellung des Kegels

Dann wird jede Aufnahme *i* der *N* Aufnahmen folgenderweise verarbeitet:

- mittels Hough-Transformation werden die drei Geraden a_i, b_i und c_i detektiert, die den drei Seiten des Kegels entsprechen,
- die Bildkoordinaten (x_i, y_i) der Kegelspitze P_i werden als Schnittpunkt der Geraden a_i und b_i berechnet und gespeichert,
- Parameter (r_i, θ_i) der Gerade c_i werden ebenfalls gespeichert.

Abbildung 4.2 veranschaulicht diese Vorgehensweise.

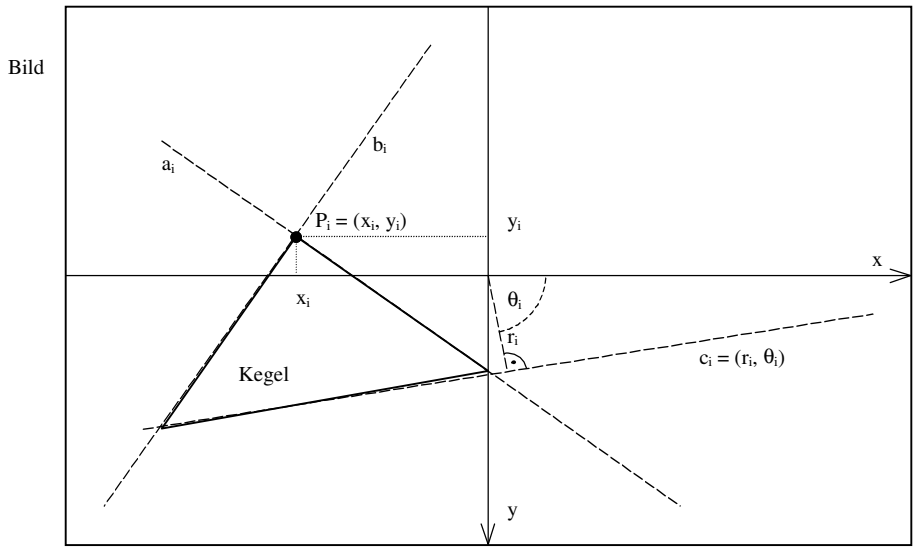


Abbildung 4.2: Verarbeitung eines Eingabebildes

Nachdem alle *N* Eingabebilder auf oben beschriebene Art und Weise bearbeitet sind, wird der arithmetische *x*-, *y*-, *r*- und θ- Mittelwert ermittelt:

$$x_m = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad y_m = \frac{1}{N}\sum_{i=1}^{N} y_i \qquad r_m = \frac{1}{N}\sum_{i=1}^{N} r_i \qquad \theta_m = \frac{1}{N}\sum_{i=1}^{N} \theta_i$$

Jetzt kann man (x_m, y_m) als Bildkoordinaten des Punktes P_m auffassen, der der Spitze des Kegels entspricht, wenn dieser genau auf die Mitte des Drehtellers gestellt wird. D.h., die Rotationsachse des Drehtellers geht durch diesen Punkt und sie ist orthogonal zur Gerade c_m, die durch Parameter (r_m, θ_m) definiert ist. Die Gerade c_m entspricht der Drehtellerebene, die von Kamera aus gesehen als eine Gerade aufgefaßt wird, wenn die optische Achse der Kamera in der Drehtellerebene liegt. Deswegen ist diese Anforderung wichtig für die Genauigkeit des Verfahrens.

Somit liegen genügend viele Informationen vor, um die Parameter der Rotationsachse zu bestimmen. Bezeichnen wir diese Achse mit c_{rot} und beschreiben wir sie mit den Parametern (r_{rot}, θ_{rot}). Aus $P_m = (x_m, y_m)$, $c_m = (r_m, \theta_m)$ und obigen Überlegungen folgt:

$$r_{rot} = x_m \sin\theta_m - y_m \cos\theta_m$$

$$\theta_{rot} = \theta_m - \frac{\pi}{2}$$

Wenn dadurch θ_{rot} kleiner als 0 wurde, also aus dem zulässigen Bereich [0, π) für die Hessesche Darstellung einer Gerade liegt, man kann dies wie folgt korrigieren:

$$\theta_{rot} = \theta_{rot} + \pi$$
$$r_{rot} = - r_{rot}$$

Somit ist die Bestimmung der Rotationsachse abgeschlossen. Abbildung 4.3 skizziert die Beziehungen zwischen den Parametern der Rotationsachse (r_{rot}, θ_{rot}) und den berechneten Mittelwerten x_m, y_m, r_m und θ_m

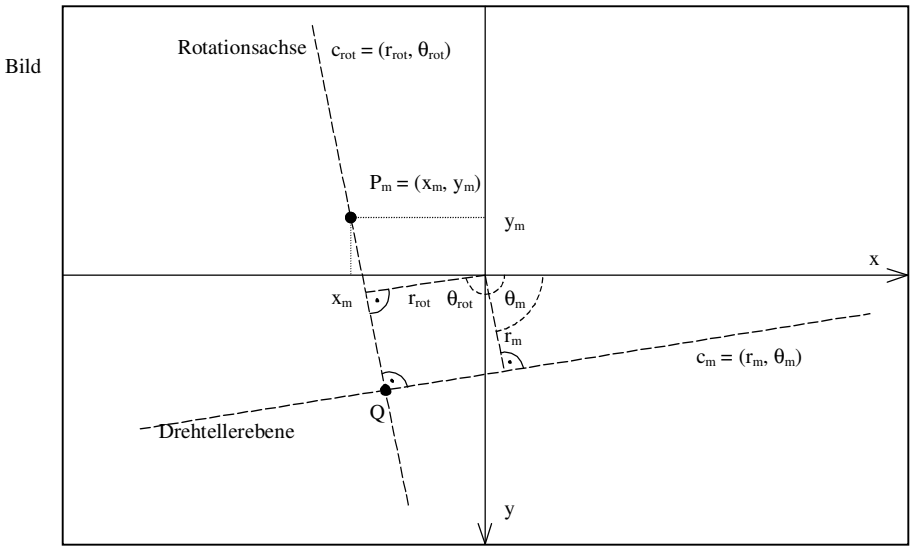


Abbildung 4.3: Berechnung der Parameter der Rotationsachse

Dann werden die Parameter der Transformation berechnet, die das Bildkoordinatensystem in das Objektkoordinatensystem umwandelt, wobei angenommen wird, daß der Koordinatenursprung des Bildkoordinatensystems in der Mitte des Eingabebildes liegt, und der Ursprung des Objektkoordinatensystems im Schnittpunkt der Drehtellerebene und der Rotationsachse, der durch den Schnittpunkt Q der Geraden c_m und c_{rot} approximiert wird. Abbildung 4.4 zeigt die geometrische Anordnung der Achsen der beiden Koordinatensysteme. Wie man in der Abbildung sehen kann, ist die y-Achse des Objektkoordinatensystems die Rotationsachse des Drehtellers. Eine weitere Annahme des Kalibrierungsverfahrens ist, daß die *xy*-Ebene des Objektkoordinatensystems parallel, bzw. seine *z*-Achse orthogonal zur Bildebene ist. Diese Annahme vereinfacht die weitere Berechnung der Parameter für die Transformation des einen Koordinatensystems in das andere.

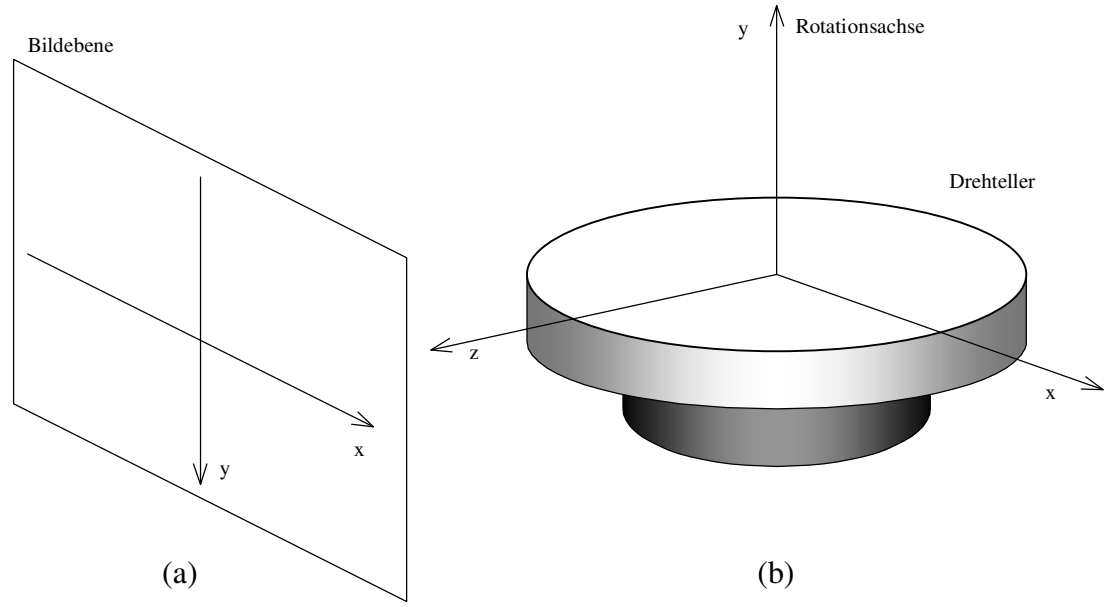


Abbildung 4.4: Geometrische Anordnung der Achsen des Bild- (a) und Objekt- (b) Koordinatensystems

4.2 Bestimmung der Transformationsmatrix

Die Transformation des Bildkoordinatensystems in das Objektkoordinatensystem kann durch affine Transformationen [Wat95] beschrieben werden, bzw. durch eine Zusammensetzung von Rotation, Skalierung und Translation. Wenn man die Koordinaten eines Punktes im Objekt- (Welt-) Koordinatensystem mit (x_w, y_w, z_w) bezeichnet, die Koordinaten des Bildkoordinatensystem mit (x_b, y_b, z_b), und werden alle Koordinaten in ihrer homogenen Form geschrieben (damit auch Translation in Matrizenform dargestellt werden kann), dann gilt:

$$\begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} = S \cdot T \cdot R \cdot \begin{pmatrix} x_b \\ y_b \\ z_b \\ 1 \end{pmatrix}$$

Dabei bezeichnet S die Skalierungs-, R die Rotations- und T die Translationsmatrix. Ist der Rotationswinkel um die x-Achse α, um die y-Achse β und um die z-Achse γ, so kann die Rotationsmatrix durch die Multiplikation der Rotationsmatrizen Rx(α), Ry(β) und Rz(γ) gebildet werden, die die Rotationen um entsprechende Achsen beschreiben.

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Daraus folgt

$$R = R_x(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma) =$$

$$= \begin{pmatrix} \cos\beta\cos\gamma & -\cos\alpha\sin\gamma + \sin\alpha\sin\beta\cos\gamma & \sin\alpha\sin\gamma + \cos\alpha\sin\beta\cos\gamma & 0 \\ \cos\beta\sin\gamma & \cos\alpha\cos\gamma + \sin\alpha\sin\beta\sin\gamma & -\sin\alpha\cos\gamma + \cos\alpha\sin\beta\sin\gamma & 0 \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Skalierungsmatrix S und Translationsmatrix T sind gegeben durch:

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Aus der geometrischen Anordnung der Achsen (siehe Abbildung 4.4) der Koordinatensysteme und der Annahme, daß die *xy*-Ebenen der beiden Koordinatensysteme parallel sind, sowie der Bildkoordinaten (x_Q, y_Q) des Koordinatenursprungs Q des Objektkoordinatensystems und der Parameter (r_{rot}, θ_{rot}) der Rotationsachse c_{rot} (Abbildung 4.3) des Drehtellers folgt:

$$\alpha = 0 \qquad t_x = -x_Q$$
$$\beta = 0 \qquad t_y = -y_Q$$
$$\gamma = -\theta_{rot} \qquad t_z = 0$$

t_z kann eine beliebige Zahl sein, da die *z*-Achse des Objektkoordinatensystems orthogonal zur Bildebene ist, was heißt, daß die Bildkoordinaten eines Punktes nichts über die *z*-Koordinate im Objektkoordinatensystem sagen.

Aufgrund der bekannten Höhe H des Kegels, also seiner Höhe im Objektkoordinatensystem, und seiner Pixelhöhe P_mQ im Bildoordinatensystem (Abbildung 4.3), kann man die Parameter s_x, s_y und s_z der Skalierungsmatrix berechnen:

$$s_x = s_y = s_z = \frac{H}{PQ}$$

Bezeichnen wir diese Größe mit k. Jetzt können wir die einzelnen Transformationsmatrizen S, T und R ausmultiplizieren und somit erhalten wir die gesuchte Transformationsmatrix, die das Bildkoordinatensystem in das Objektkoordinatensystem umwandelt:

$$S \cdot T \cdot R = \begin{pmatrix} k & 0 & 0 & 0 \\ 0 & -k & 0 & 0 \\ 0 & 0 & k & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -x_Q \\ 0 & 1 & 0 & -y_Q \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} k\cos\gamma & -k\sin\gamma & 0 & -k\,x_Q \\ -k\sin\gamma & -k\cos\gamma & 0 & k\,y_Q \\ 0 & 0 & k & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4.3 Bestimmung der Entfernung zwischen der Kamera und der Rotationsachse

Für die Bestimmung der Entfernung zwischen dem optischen Zentrum der Kamera und der Rotationsachse des Drehtellers sind folgende Parameter notwendig:

- die fokale Länge *f* der Kamera,
- die Höhe *d* eines Pixels in mm,
- die reale Höhe *H* des Kegels (bzw. seine Höhe im Objektkoordinatensystem),
- die Pixelhöhe *h* des Kegels (bzw. seine Höhe im Bildkoordinatensystem)

Die ersten zwei Parameter sind innere Parameter der Kamera und sie werden aus einem File abgelesen, der dritte wird als Inputparameter dem Task übergeben wird und der vierte wird durch den Task berechnet (*h* ist die Linie P_mQ in der Abbildung 4.4).

Die gesuchte Entfernung *D* wird folgenderweise berechnet:

$$D = \frac{H \cdot f}{h \cdot d}$$

Diese Gleichung folgt aus in der Abbildung 4.5 veranschaulichten Beziehung:

$$D : H = f : (h \cdot d)$$

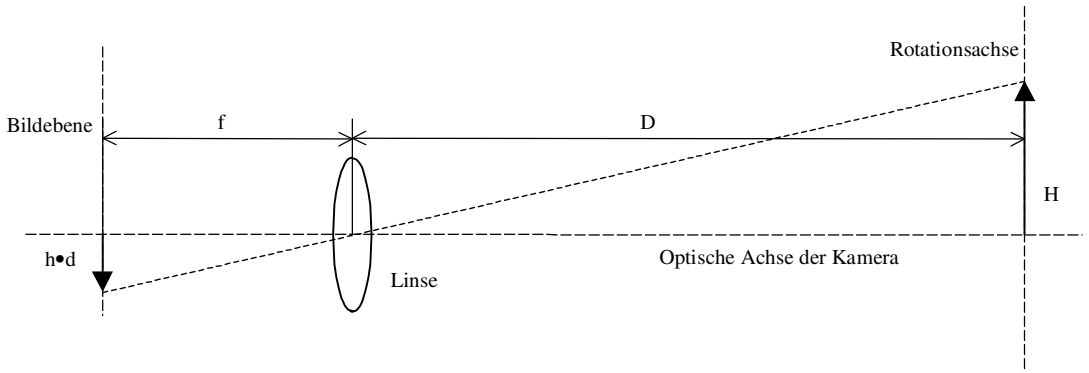


Abbildung 4.5: Berechnung der Entfernung zwischen der Kamera und der Rotationsachse

5. Tests und Ergebnisse

Hier werden die durchgeführten Test zusammengefaßt und die Ergebnisse analysiert, im ersten Teil bezüglich der linearen Hough-Transformation und im zweiten bezüglich der Drehtellerkalibrierung.

5.1 Lineare Hough-Transformation

Im ersten Beispiel wird ein künstlich erzeugtes Bild (Abbildung 5.1) als Eingabebild genommen. An diesem Beispiel können Stärken und Schwächen des Algorithmus gezeigt werden.

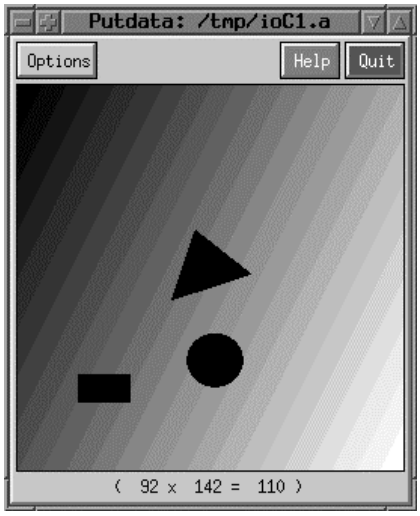
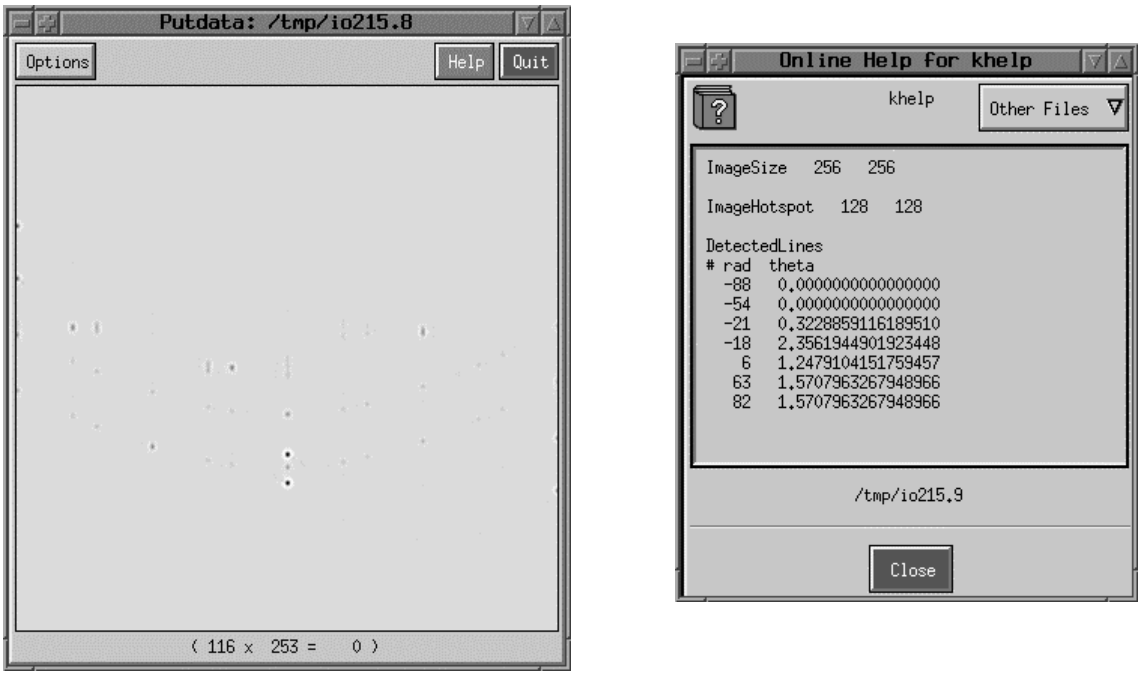


Abbildung 5.1: Das erste Eingabebild

Auf dieses Eingabebild wird die lineare Hough-Transformation zweimal angewendet, mit fast gleichen Parametern. In beiden Fällen wird der Sobel-Operator als Gradientenoperator verwendet, sowie ein Laplacian-of-Gaussian Kernel der Größe 11x11 und der Standardabweichung σ=1.4 für die Akkumulatorsäuberung, wobei der Akkumulator 360 Pixel breit wird und sein Schwellwert auf 40% der (absolut) höchsten Akkumulatorwert gesetzt wird. Der Unterschied liegt im Gradientenbetragsschwellwert, der im ersten Fall 30% und im zweiten 0% des (absolut) höchsten Werts im Gradientenbetragsbild beträgt. Für die genaue Bedeutung aller Parameter siehe Anhang B, Abschnitt B.3.3.

Im ersten Fall (Schwellwert des Gradientenbetrags = 30%) liefert die lineare Hough-Transformation den Akkumulator und die ASCII-Datei aus der Abbildung 5.2:



(a) (b)

Abbildung 5.2: Akkumulator (a) und ASCII–Ausgabedatei (b) des ersten Tests

Wenn die ASCII–Datei aus der Abbildung 5.2 in ein Binärbild umgewandelt wird (mittels `TosLinesToImage` Tasks, siehe Anhang B) und dieses mit dem Eingabebild überlagert wird, kann man sich die gefundenen Geraden veranschaulichen, was in der Abbildung 5.3 gezeigt ist:

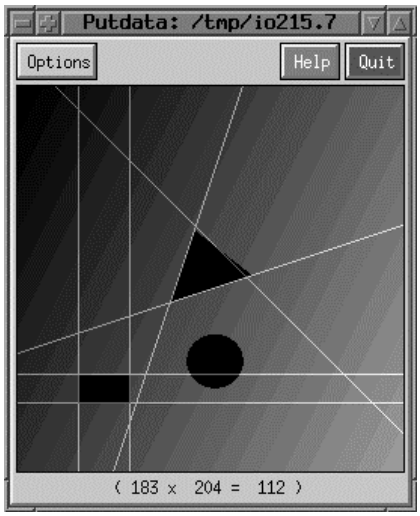
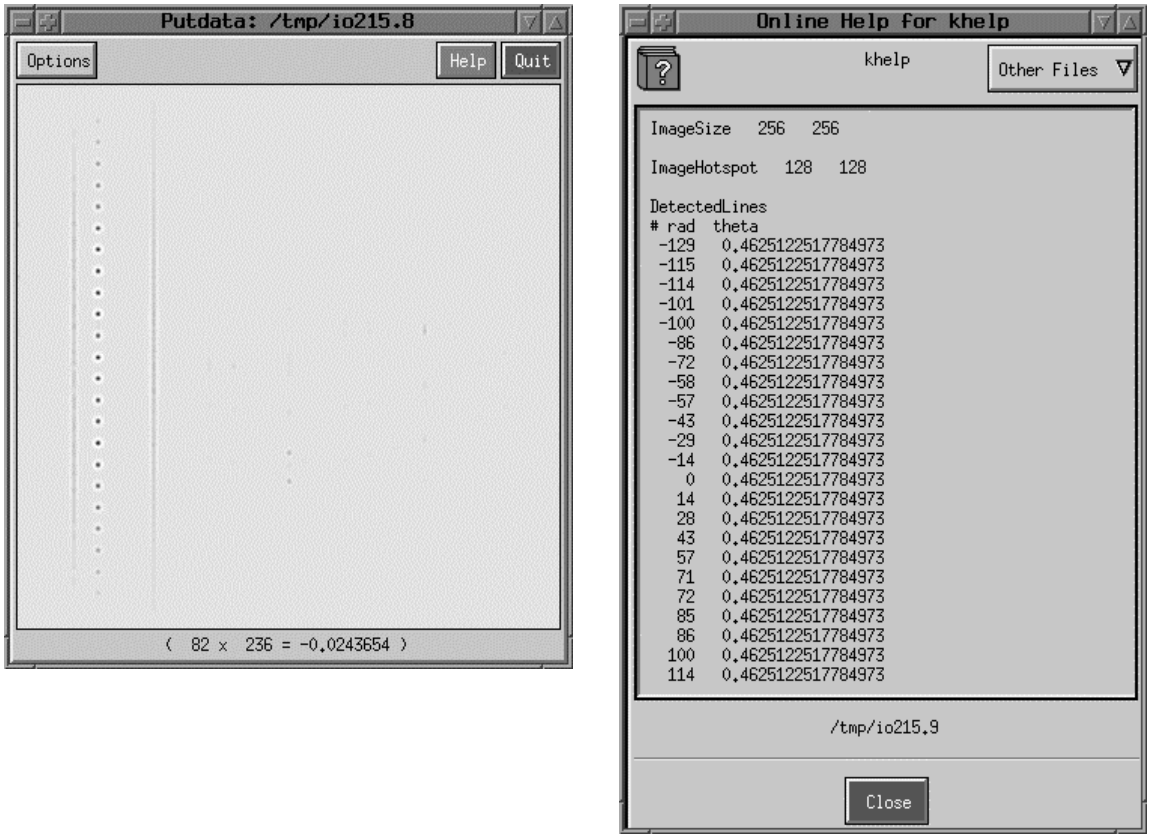


Abbildung 5.3: Gefundene Geraden im ersten Test

Die Ergebnisse für den zweiten Fall (Schwellwert des Gradientenbetrags = 0%) sind in den Abbildungen 5.4 und 5.5 gezeigt.



(a) (b)

Abbildung 5.4: Akkumulator (a) und ASCII–Ausgabedatei (b) des zweiten Tests

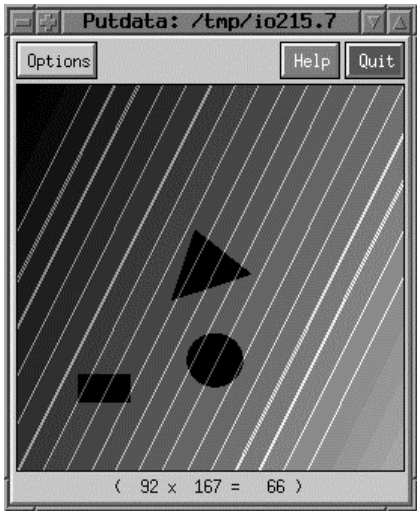


Abbildung 5.5: Gefundene Geraden im zweiten Test

Was diese Beispiele zeigen ist, daß sich durch eine geeignete Parametrisierung der linearen Hough-Transformation „interessante“ von „uninteressanten“ Geraden unterscheiden lassen, wobei es vor allem zwischen „starken“, also Linien mit einem großen Gradientenbetrag, und langen Linien unterschieden werden kann. Anhand dieser Beispiele kann man auch sehen, daß die Teile des Bildes, die keine geradlinige Struktur aufweisen (wie der Kreis in diesem Beispiel) keinen großen oder überhaupt keinen Einfluß auf das Ergebnis der linearen Hough-Transformation haben.

Das Ergebnis des ersten Tests (Abbildung 5.3) zeigt auch eine Schwäche des Algorithmus für künstliche Bilder. Man sieht, daß eine Seite des Dreiecks falsch detektiert wurde. Wo liegt das Problem? Durch Diskretisierung dieser Seite des Dreiecks, die einen Gradientenrichtung knapp unter $-\pi/4$ haben sollte, besteht sie eher aus einer Vielzahl der Geradenstücken mit der Gradientenrichtung genau $-\pi/4$. Dadurch erhalten fast alle Punkte dieser Linie im Gradientenrichtungsbild den Wert $-\pi/4$, bzw. diese Linie wird im Akkumulator durch einen Cluster von Punkten mit $\theta = 3\pi/4$ dargestellt und einer dieser Punkte wird als repräsentativ für diese Linie betrachtet. Dieses Phänomen ist bei realen Bildern nicht zu erwarten, da dort die stufenförmige Form einer Linie durch verschiedene Grauwerte der Pixel, aus der die Linie besteht, mindestens zum Teil ausgeglichen wird.

Im nächsten Beispiel wird ein reales Bild (Abbildung 5.6) als Eingabebild genommen. Die Ergebnisse sind in der Abbildung 5.7 zusammengefaßt, wobei wieder ein Laplacian-of-Gaussian Kernel der Größe 11x11 mit σ=1.4 für die Säuberung des Akkumulators verwendet wurde und die Schwellwerte so gewählt wurden, daß die vier bedeutsamsten Geraden detektiert werden. Abbildung 5.8 veranschaulicht zusätzlich das Ergebnis wenn der Akkumulator überhaupt nicht gesäubert wird, und zeigt somit wie sich die einzelnen Linien im Eingabebild auf eine ganze Menge von mathematischen Geraden abbilden.

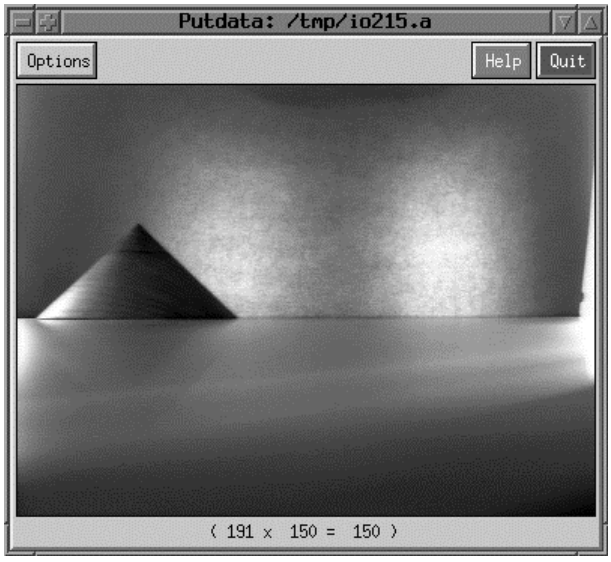
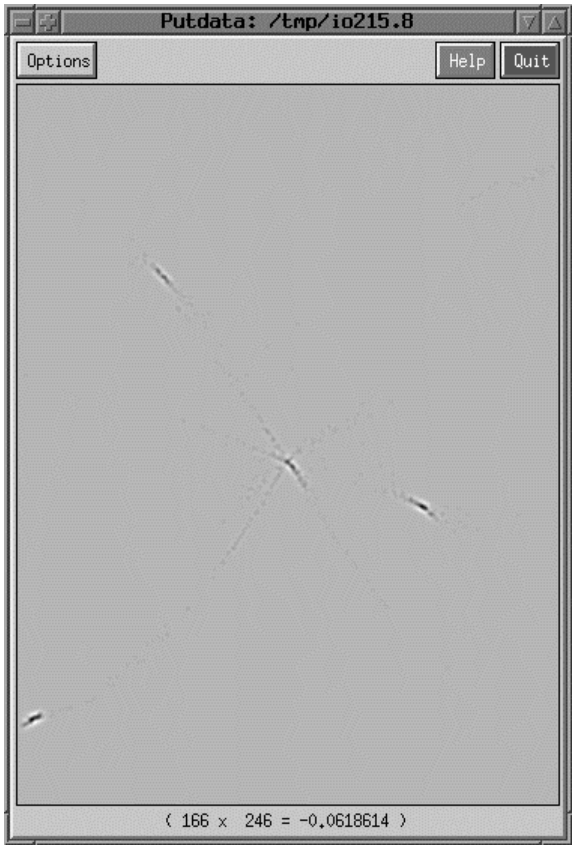
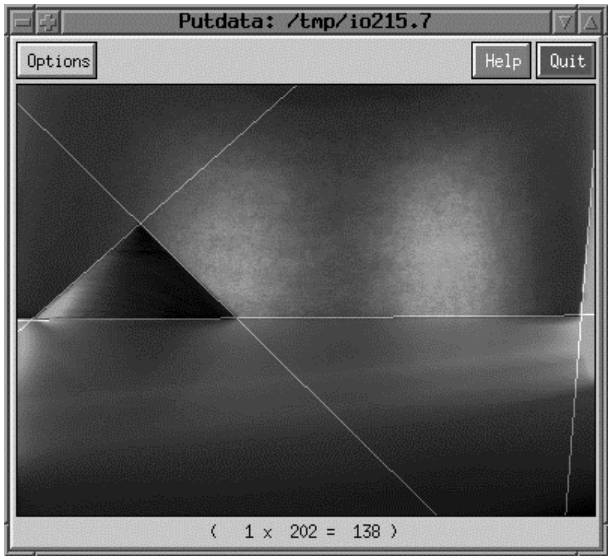


Abbildung 5.6: Das zweite Eingabebild



(a)



(b)

Abbildung 5.7: Akkumulator (a) und die gefundenen Geraden (b) im zweiten Eingabebild

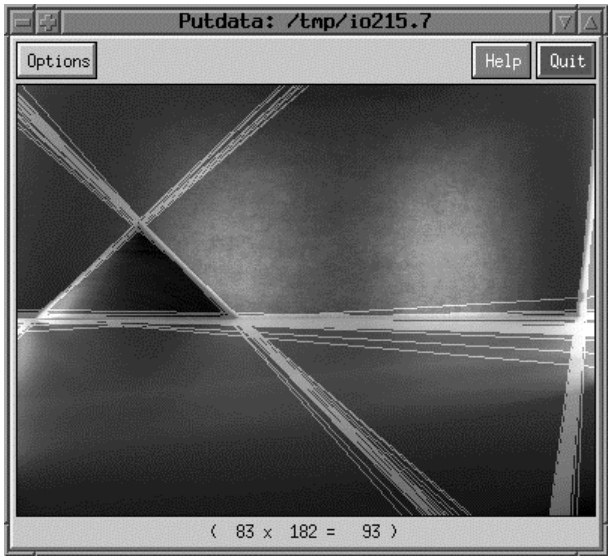
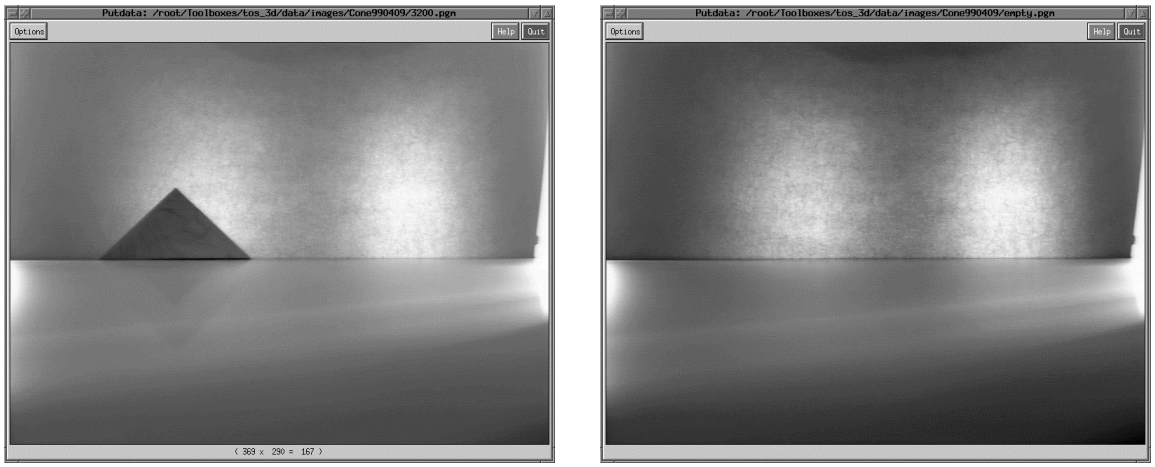


Abbildung 5.8: Die gefundenen Geraden im zweiten Eingabebild, ohne Akkumulatorsäuberung

Hier wird noch etwas über den Aufwand der linearen Hough-Transformation gesagt. Bei einem Eingabebild der Größe 512x512 Pixel, Akkumulatorbreite 180 Pixel, unter Verwendung der Sobel-3x3-Kernels für den Gradientenoperator und eines 11x11 Laplacian-of-Gaussian Kernels für die Akkumulatorsäuberung betrug die Rechenzeit auf einem Pentium 133MHz Rechner mit 32MB RAM ca. 22 Sekunden. Der Großteil der CPU-Zeit liegt bei der Akkumulatorsäuberung, besonders wenn ein großer Kernel dafür verwendet wird. Eine Verdoppelung der Länge und der Breite des Eingabebildes (also eine Vervierfachung der Pixelanzahl) führt zur Vervierfachung der Rechenzeit, ebenfalls eine Verdoppelung der Größe des LoG-Kernels, während die Verdoppelung der Akkumulatorbreite zur Verdoppelung der Rechenzeit führt.

5.2 Drehtellerkalibrierung

Für das Testen der Drehtellerkalibrierung wurden insgesamt 36 Aufnahmen des Kegels aus verschiedenen Blickwinkeln gemacht, plus eine Aufnahme des „leeren“ Drehtellers. Abbildung 5.9 zeigt eine der Eingabeaufnahmen und die des leeren Drehtellers.

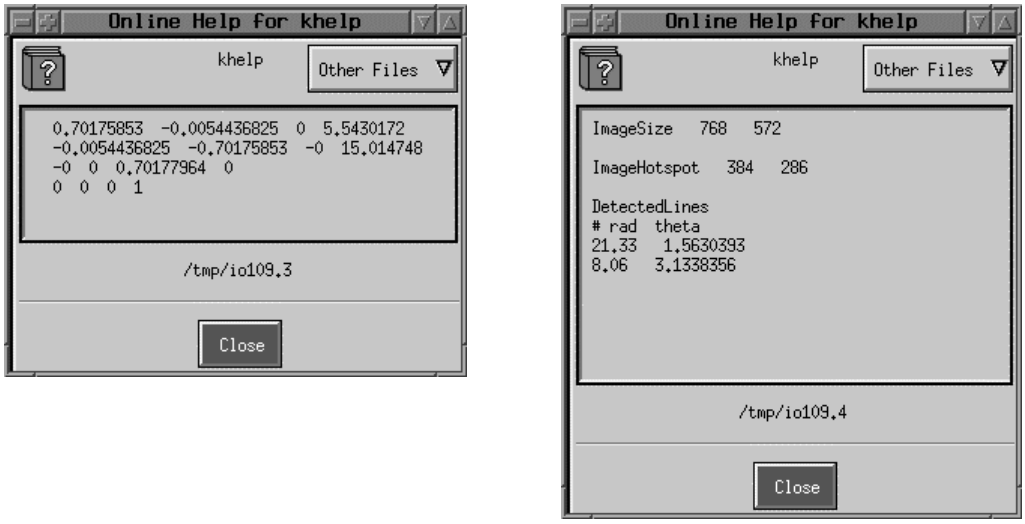


(a) (b)

Abbildung 5.9: Eine der Aufnahmen des Kegels (a) und der leere Drehteller (b)

Der Task zur Drehtellerkalibrierung, `TosCalib`, wurde für verschiedene Untermengen dieser Serie von Aufnahmen ausgeführt – einmal für alle 36 Bilder, einmal für 4 und einmal für nur 2 Bilder. Für alle Tests wurden die Parameter für die lineare Hough-Transformation gleich gesetzt – Schwellwert des Gradientenbetrags auf 50%, Akkumulatorbreite auf 180 Pixel, wobei der Sobel-Operator als Gradientenoperator und Laplacian-of-Gaussian Kernel der Größe 11 und der Standardabweichung σ=1.4 für Akkumulatorsäuberung verwendet wurden.

Im ersten Fall, wo alle Aufnahmen zur Kalibrierung herangezogen wurden, lieferte `TosCalib` die Dateien, die der Abbildung 5.10 zu entnehmen sind.



(a) (b)

Abbildung 5.10: Berechnete Transformationsmatrix (a) und Parameter der Achsen des Objektkoordinatensystems (b)

Abbildung 5.10 (a) zeigt die berechnete Transformationsmatrix. Das ist die einzige Ausgabedatei, die an die Tasks, die dieses Kalibrierungsverfahren verwenden, weitergeleitet werden soll. In der Abbildung 5.10 (b) sind die Parameter der Achsen des Objektkoordinatensystems zu sehen, bezüglich des Bildkoordinatensystems. Diese Datei kann mittels des Tasks `TosLinesToImage` (siehe Anhang B) in ein Binärbild umgewandelt werden, das als eine visuelle Überprüfung des Ergebnisses dienen kann. Wird die Datei aus der Abbildung 5.10 (b) in ein Binärbild umgewandelt, und dieses mit einer Aufnahme überlagert, in der der Kegel ungefähr auf dem Rotationszentrum des Drehtellers liegt, entsteht das Bild aus der Abbildung 5.11.

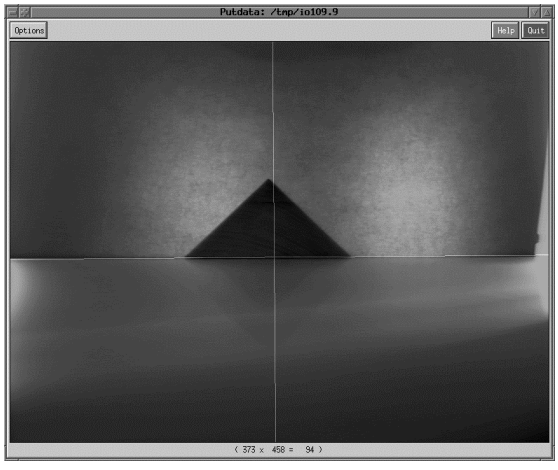
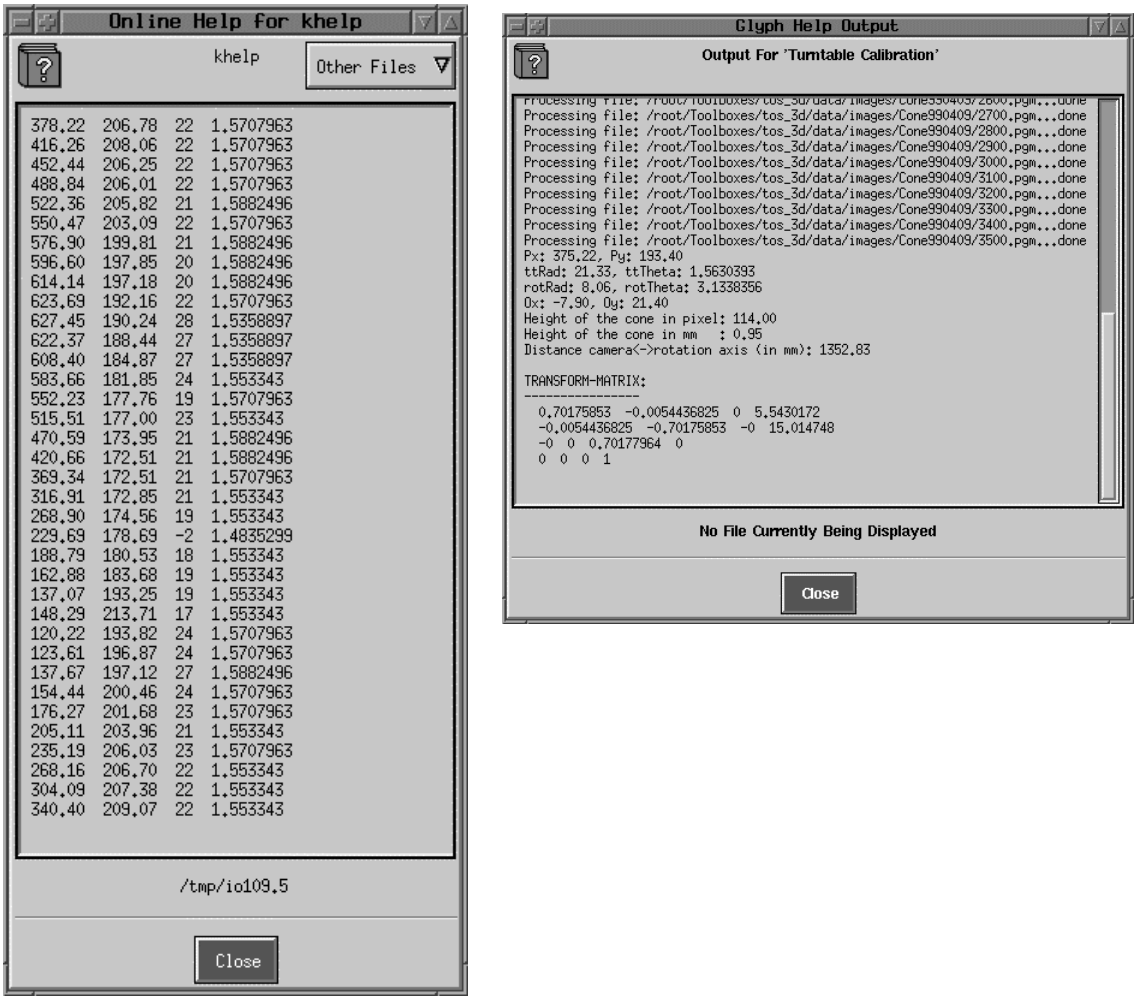


Abbildung 5.11: Achsen des Objektkoordinatensystems

Die dritte (und die letzte) Ausgabedatei, die in der Abbildung 5.12 (a) zu sehen ist, dient auch zur Überprüfung des Ergebnisses. Sie wird auch intern verwendet, für die Berechnung des

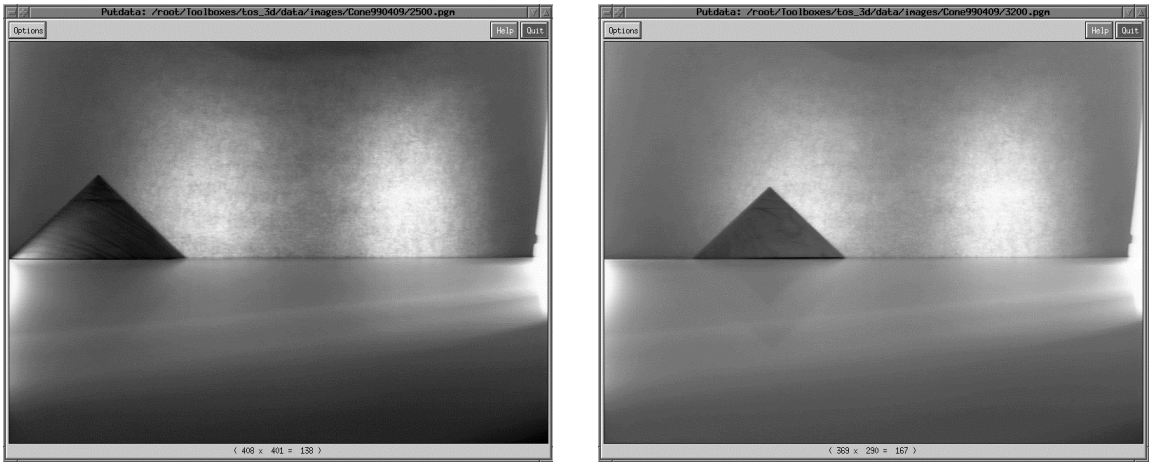
Mittelwertes jeder Spalte. Die ersten zwei Spalten sind *x*- und *y*-Koordinaten der Kegelspitze im jeweiligen Eingabebild, die dritte und die vierte der Normalabstand *r* von der Mitte des Eingabebildes und der zugehörige Winkel θ der detektierten Gerade, auf der die untere Seite des Kegels liegt. Abbildung 5.12 (b) zeigt Info-Ausgabe auf `kstderr`.



(a) (b)

Abbildung 5.12: Testausgabedatei (a) und Ausgabe auf `kstderr` (b)

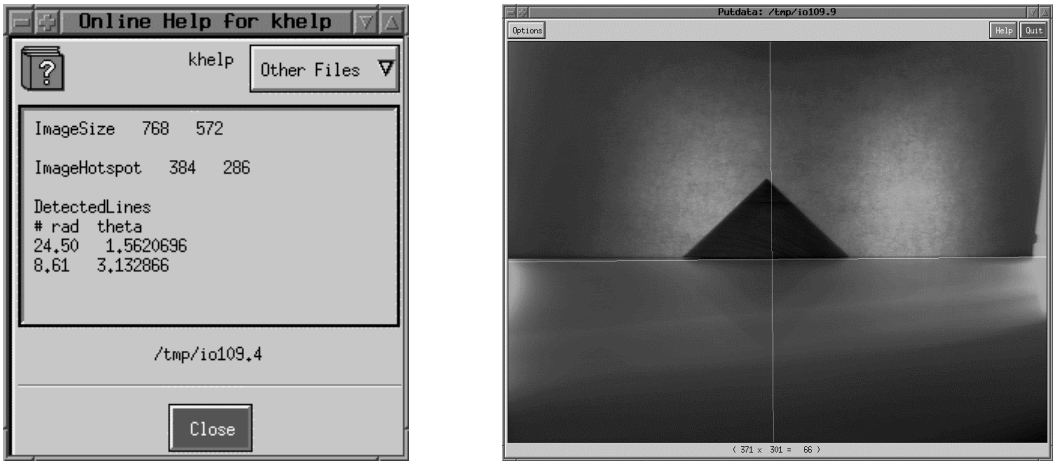
Wird ein Eingabebild mittels Khoros Tasks `putdata` geöffnet, kann händisch überprüft werden, wie genau die Kegelspitzen detektiert wurden. In diesem Test, für alle der 36 Aufnahmen, bis auf 4 oder 5, war die Distanz zwischen der berechneten und der händisch festgestellten Kegelspitze kleiner oder gleich 3 Pixel, für den Großteil 1 bis 2 Pixel. Von den anderen Bildern war der Fehler am größten für das Bild aus der Abbildung 5.13 (a): dort war der Fehler in x-Richtung 20 und in y-Richtung 26 Pixel! Die Ursache war (genau sowie bei anderen Bildern, wo der Fehler groß war) schlechte Beleuchtung – auf dem Kegel sind Schatten entstanden, was zur Detektion von nicht existierenden Linien führte. Bei den Aufnahmen, wo der Kegel gleichmäßig beleuchtet war (Abbildung 5.13 (b)), war die Detektion der Spitze auf 1 Pixel genau.



(a) (b)

Abbildung 5.13: Ein schlechtes (a) und ein gutes (b) Eingabebild

Im zweiten Test wurden 4 der „guten“ Eingabebilder zur Kalibrierung herangezogen, mit dem Winkel von 90° zwischen zwei benachbarten Bildern. Das Ergebnis ist in den Abbildungen 5.14 und 5.15 zusammengefaßt. Die Ausgabedatei mit Transformationsmatrix ist nicht gezeigt, da diese Matrix auch in der Info-Ausgabe in der Abbildung 5.15 zu sehen ist.



(a) (b)

Abbildung 5.14: Parameter der Achsen des Objektkoordinatensystems (a) und das entsprechende Testbild (b) des Tests mit 4 Aufnahmen

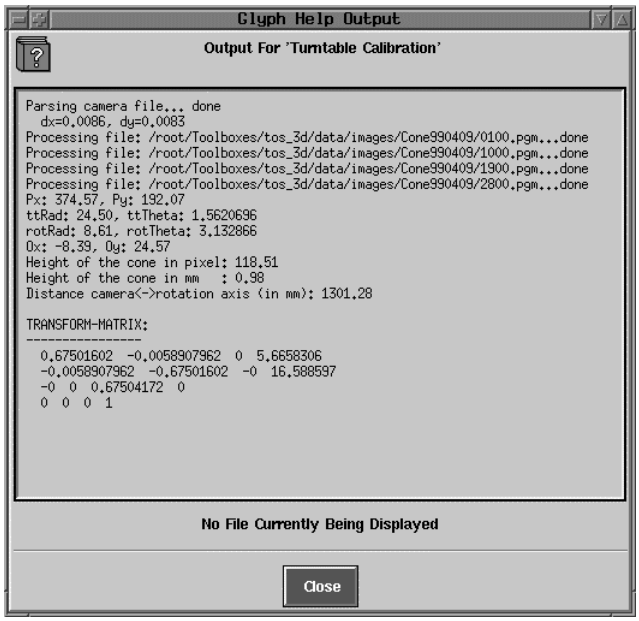
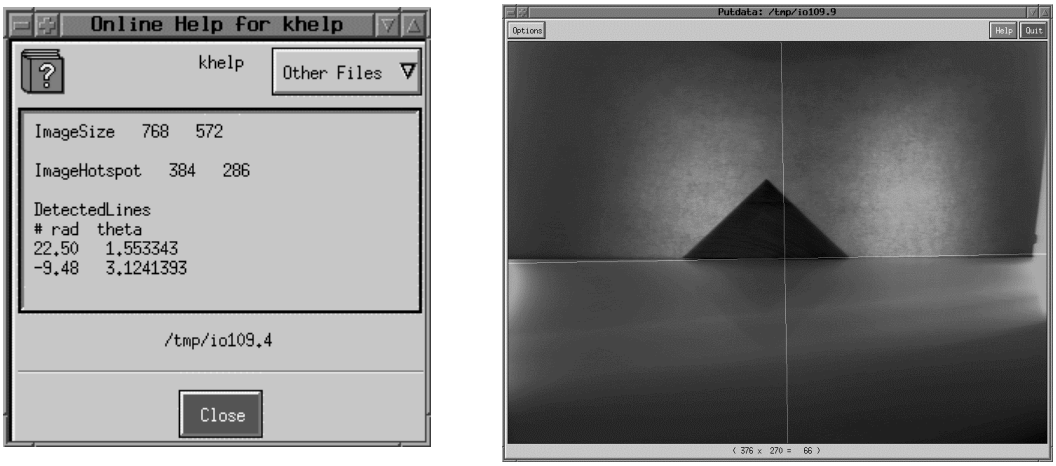


Abbildung 5.15: Ausgabe des Tests mit 4 Aufnahmen auf `kstderr`

Im letzten Test wurden nur zwei aus der Gruppe der „guten“ Aufnahmen für die Kalibrierung verwendet. Der Winkel zwischen diesen Aufnahmen betrug 180°. Die Abbildung 5.15 faßt die Ergebnisse zusammen.



(a) (b)

Abbildung 5.16: Parameter der Achsen des Objektkoordinatensystems (a) und das entsprechende Testbild (b) des Tests mit 2 Aufnahmen

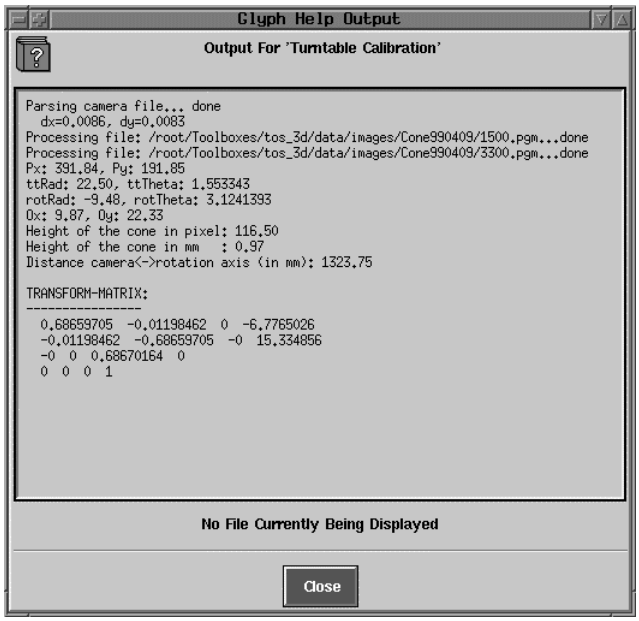


Abbildung 5.17: Ausgabe des Tests mit 2 Aufnahmen auf `kstderr`

Wenn die drei Testbilder (mit Achsen des Objektkoordinatensystems) aus den Abbildungen 5.11, 5.14 (b) und 5.16 (b) übereinander gelegt, entsteht das Bild aus der Abbildung 5.18 (hier vergrößert, damit man die berechneten Achsen deutlicher sehen kann).

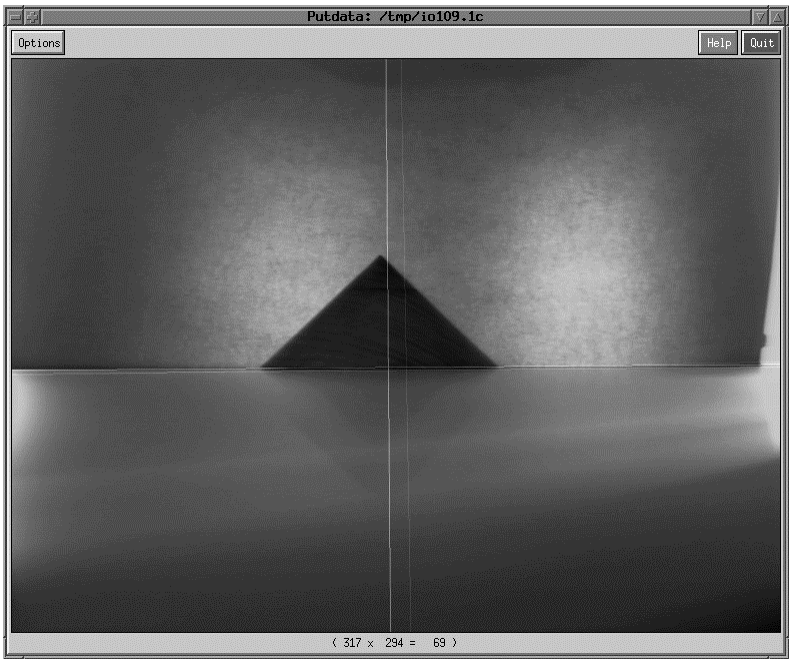


Abbildung 5.18: Die drei Testausgabebilder, übereinander gelegt

Die linke vertikale Gerade in der Abbildung 5.18 entspricht der berechneten Rotationsachse sowohl für den Test mit 36, als auch für den mit 4 Eingabebildern und die rechte vertikale Gerade der Rotationsachse für den Test mit 2 Eingabebildern. Bei der horizontalen Achse sind auch kleine Unterschiede zwischen den Ergebnissen der drei Tests zu bemerken. Die Genauigkeit der Berechnung der horizontalen Achse bei den einzelnen Tests sieht man aber deutlicher in den Abbildungen 5.11, 5.14 (b) und 5.16 (b).

Im ersten Tests (36 Eingabeaufnahmen) betrug die berechnete Entfernung zwischen der Kamera und der Rotationsachse des Drehtellers ca. 135.28 *cm*, im zweiten (4 Eingabeaufnahmen) 130.13 *cm* und im dritten (2 Eingabeaufnahmen) 132.38 *cm*.

Für die Bestimmung der Genauigkeit des Verfahrens wurden 4 Durchläufe des Verfahrens gemacht, jeweils mit 6 „guten“ Eingabebildern. Die maximale Abweichung des Normalabstands der Rotationsachse des Drehtellers bezüglich des Bildhauptpunktes vom Mittelwert war 1.2 Pixel, die maximale Abweichung des zugehörigen Winkels 0.00435 rad (0.12°) und die maximale Abweichung der Entfernung zwischen der Kamera und der Rotationsachse 10.7 mm (der Mittelwert war 133.62 cm).

Was diese Ergebnisse zeigen ist, daß, zumindest visuell, schon mit einer kleinen Anzahl von Eingabebildern die zu berechnenden Parameter (Parameter der Achsen des Objektkoordinatensystems bezüglich des Bildkoordinatensystems und die Entfernung zwischen der Kamera und der Rotationsachse) näherungsweise bestimmt werden können. Die steigende Anzahl von Eingabebildern erhöht die Genauigkeit der Berechnungen, es macht aber Sinn, die Genauigkeit der Detektion der Kegelspitze händisch zu Überprüfen, damit man feststellen kann, ob einige der Eingabebilder zu einer starken Verfälschung der Ergebnisse konnten geführt haben, und ggf. den Kalibriervorgang wiederholen.

6. Zusammenfassung und Ausblicke

In dieser Arbeit wurde ein Verfahren zur Kalibrierung eines Aufnahmesystems beschrieben, das aus einer Kamera und einem Drehteller besteht, wobei die optische Achse der Kamera in der Drehtellerebene liegt und orthogonal zu seiner Rotationsachse ist. Es wurde auch die dem Verfahren zugrundeliegende lineare Hough-Transformation ausführlich beschrieben.

Die Tests der linearen Hough-Transformation ergaben, daß sie ein robustes Verfahren für Detektion der Geraden im Eingabebild ist. Wenn das Eingabebild tatsächlich gerade Linien enthält, kann durch geeignete Parameterangabe hohe Genauigkeit erzielt werden, wobei sich auch interessante Geraden des Eingabebildes von uninteressanten unterscheiden lassen. Die im Rahmen dieses Praktikums implementierte lineare Hough-Transformation könnte verbessert werden, indem nach der Gradientenoperation eine Verdünnung der Kanten durchgeführt wird, bevor es mit den Hough-Transformation angefangen wird. Es wurde auch keine Tracking (Kantenverfolgung) implementiert, die feststellen würde, wo der Anfang und wo das Ende der Linien ist, die auf den detektierten Geraden liegen – es werden nur die mathematischen Parameter dieser Geraden bestimmt.

Das Kalibrierungsverfahren beruht auf der Verfolgung der Spitze des Kegels, der als Kalibrierobjekt verwendet wird. Der Erfolg des Verfahrens hängt sehr stark von der Qualität der Aufnahmen ab, da unerwünschte Schatten auf dem Kegel zur fehlerhaften Detektion seiner Spitze führen können. Das Verfahren macht auch einige vereinfachende Annahmen, die seine Genauigkeit vermindern – es wurde vom idealen Kameramodell ausgegangen, ohne Linsenverzerrung, mit dem Bildhauptpunkt genau in der Mitte des aufgenommenen Bildes und es wurde angenommen, daß die optische Achse der Kamera genau in der Drehtellerebene liegt und daß sie orthogonal zur Rotationsachse des Drehtellers ist. Ob unter diesen Annahmen das Verfahren ausreichend genaue Ergebnisse liefert, wird sich im Task „Shape from Silhouette“ zeigen, für den dieses Verfahren entwickelt wurde. Die Realisierung dieses Task steht noch bevor, daher könnte es noch zu kleinen Veränderungen des hier beschriebenen Kalibrierungsverfahrens kommen.

7. Literaturverzeichnis

[Bäs93] H. Bässmann and Ph. W. Besslich. *Bildverarbeiung Ad Oculos*. Springer-Verlag Berlin Heidelberg New York, 1993

[Dud72] R. O. Duda and P. E. Hart. Using the Hough transforms to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11-15, 1972

[Har91] Robert M. Haralick and Linda G. Shapiro. Glossary of computer vision terms. *Pattern Recognition*, 24(1):69-93, 1991

[Hou62] P. V. C. Hough. *A method and means for recognizing complex patterns*. U.S., Patent 3,069,654, 1962

[Khoros] *www.khoral.com*

[Linux] *www.linux.org*

[Son93] Milan Sonka, Vaclav Hlavac and Roger Boyle. *Image processing, analysis and machine vision*. Chapman & Hall Computing, 1993

[Tsa86] Roger Y. Tsai. An efficient and accurate camera calibration technique for 3D machine vision. *Proceedings of IEEE conference on computer vision and pattern recognition*, pages 364-374, Miami Beach, FL, 1986

[Wat95] Alan Watt. *3D computer graphics*. Addison-Wesley, 1995

Anhang A: Implementierungsdetails

Hier wird auf die Implementierungsdetails der entwickelten Tasks näher eingegangen. Erster Teil beschreibt die Routinen des Tasks zur linearen Hough-Transformation und der zweite die Routinen des Tasks zur Drehtellerkalibrierung. Alle Tasks wurden in der Programmiersprache *C* geschrieben, unter Betriebssystem *Linux* 2.0.29 und mit Hilfe der *Khoros 2.2* Softwareentwicklungsumgebung. Auf der Institutshardware wurden die Tasks auf die Rechner mit Betriebssystem *SGI Irix* 6.5 und *Linux* 2.0.33 implementiert und getestet.

A.1 Lineare Hough-Transformation

Lineare Hough-Transformation wurde in zwei Varianten implementiert, als:
- *kroutine* (*Khoros*-Routine) `TosHoughTrans`, also ein ausführbares Programm, und
- *Library-Call* `lTosHoughTrans()`, das von C-Programmen aus verwendet werden kann.

A.1.1 kroutine `TosHoughTrans`

Alles was diese Routine macht ist das Ablesen der Parameter aus der Kommandozeile, Öffnen von den entsprechenden Ein- und Ausgabeobjekten, deren Einsetzen in die entsprechenden Parameter der Library-Calls `lTosHoughTrans()` und schließlich der Aufruf des Library-Calls.

A.1.2 Library-Call `lTosHoughTrans()`

Die Implementation der linearen Hough-Transformation ist in dieser Routine zu finden. Wie es im Abschnitt 3 schon erwähnt wurde, besteht sie aus 4 Teilen:
- Anwendung eines Gradientenoperators auf das Eingabebild, der als Ergebnis sowohl die Gradientenbetrag als auch die Gradientenrichtung liefert,
- Hough-Transformation,
- Akkumulatorsäuberung,
- Optionale Ausgabe der gefundenen Geraden in eine spezial formatierte Textdatei.

Es folgt eine genauere Beschreibung jedes Schrittes.

Gradientenoperator

Gradientenoperator wurde durch vier Aufrufe der Funktionen aus der DATAMANIP *Khoros* library realisiert. Zuerst werden durch die Anwendung des linearen operators `lklinearop()` auf das Eingabebild, einmal mit dem Kernel für die x- und einmal für die y-Richtung, Kanten in x- und y-Richtung detektiert, wobei die Ergebnisbilder in Variablen `dx_tmp` und `dy_tmp` gespeichert werden:

```
KCALL(lklinearop(inImage_ref, kernelX_ref,
                 0, 0.0, 0.0, hotspotX, 0, NULL, NULL, 1,
                 dxTmp));
KCALL(lklinearop(inImage_ref, kernelY_ref,
                 0, 0.0, 0.0, hotspotY, 0, NULL, NULL, 1,
                 dyTmp));
```

`inImage_ref` bezeichnet das Eingabebild, `kernelX_ref` und `kernelY_ref` den jeweils anzuwendenden Kernel, `hotspotX` und `hotspotY` die Koordinaten ihrer Mitten und `KCALL` ein im *Khoros* vordefiniertes Makro, daß im Falle des Fehlschlagens des Library-Calls eine genauere Angabe des Ortes und der Ursache des Fehlers ermöglicht. Für genaue Bedeutung aller Parameter siehe *Khoros* `lklinearop()` - Manual Page. Dann werden die beiden temporären Bilder `dxTmp` und `dyTmp` durch Anwendung zweier arithmetischen Operatoren punktweise verknüpft:

```
KCALL(lkarith2(dyTmp, dxTmp,
      0.0, 0.0, FALSE, "hypot", FALSE,
      magTmp));
KCALL(lkarith2(dyTmp, dxTmp,
      0.0, 0.0, FALSE, "atan2", FALSE,
      angTmp));
```

Im ersten Aufruf wird das Bild `magTmp` erzeugt, wobei für jedes Pixel *(x,y)* dieses Bildes gilt:

$$\mathtt{magTmp}(x, y) = \sqrt{\mathtt{dxTmp}(x, y)^2 + \mathtt{dyTmp}(x, y)^2}$$

`magTmp` ist also das Gradientenbetragsbild. Das Gradientenrichtungsbild `angTmp` wird durch den zweiten Aufruf erzeugt, wobei gilt:

$$\mathtt{angTmp}(x, y) = \arctan \frac{\mathtt{dyTmp}(x, y)}{\mathtt{dxTmp}(x, y)}$$

Hough-Transformation

Das Gradientenbetrags- und Gradientenrichtungsbild dienen als Eingabebilder für die Hough-Transformation. Im folgenden wird dieser Teil des Programms, der sich mit der Hough-Transformation beschäftigt, verständlichkeitshalber als Pseudocode geschrieben, wobei das Gradientenbetragsbild mit `GradBetrag`, Gradientenrichtungsbild mit `GradRichtung` und der Akkumulator, das Ausgabebild der Hough-Transformation, mit `Akku` bezeichnet wird. `GradBetragSchwellwert` sagt, wie hell muß ein Pixel im Gradientenbetragsbild sein, damit es als Teil einer Kante betrachtet wird.

```
Initializiere alle Pixel des Akkumulators mit 0;
for (jedes Koordinatenpaar (x,y) der Eingabebilder)
{
```

```
  if (GradBetrag(x,y) > GradBetragSchwellwert)
  {
    Lese GradRichtung(x,y) ab; wenn es kleiner als 0 ist, dann
      erhöhe es um π; /* damit es im Intervall [0,π) liegt */
    θ = (GradRichtung(x,y)/π) * Breite des Akkumulators;
    r = x*cos(GradRichtung(x,y)) + y*sin(GradRichtung(x,y));
    Erhöhe Akku(r,θ) um 1;
  }
}
```

`r` und θ werden in integer Zahlen umgewandelt bevor sie als Koordinaten des Akkumulators verwendet werden.

Akkumulatorsäuberung

Nachdem die Bildung des Akkumulators abgeschlossen ist, entspricht jede Akkumulatorzelle mit dem Wert größer Null einer im Eingabebild detektierten Gerade. Es macht aber keinen Sinn, alle Zellen mit dem Wert größer Null als eine detektierte Gerade zu betrachten, da auch ganz gerade Linien im Eingabebild wegen Störungen und wegen der Diskretisierung der Information auf mehrere geclusterte Punkte im Akkumulator abgebildet werden. Deswegen ist es sinnvoll, den Akkumulator zu bearbeiten, indem man im jeden Cluster einen repräsentativen Punkt findet, der wirklich als eine im Eingabebild befindliche Gerade betrachtet werden kann.

Hier wurde es einfach mit Anwendung eines linearen Operators realisiert, wobei als Kernel *Laplacian-of-Gaussian* (oder kurz *LoG*-) Filter der gewünschten Größe und Standardabweichung verwendet wird, es kann aber auch ein benutzerdefinierter Kernel angegeben werden. Wenn ein *LoG*-Filter verwendet wird, dann werden die Akkumulatorzellen, die sich in der Mitte eines Clusters befinden, einen großen absoluten Wert erhalten, dadurch können sie in späteren Bearbeitung sehr einfach gefunden werden.

```
  KCALL(lklinearop(outAcc1, kernelAcc_ref,
        0, 0.0, 0.0, hotspotA, 0, NULL, NULL, 0,
        outAcc2));
```

`outAcc1` ist der Eingabeakkumulator, `outAcc2` der Ausgabeakkumulator, `kernelAcc` der verwendete Kernel und `hotspotA` ein Vektor der Koordinaten der Mitte des Kernels. `outAcc2` ist dabei nicht der vom Library-Call auszugebende Akkumulator, es ist der um die Breite und Höhe des Kernels vergrößerter Akkumulator, damit durch den linearen Operator auch die besonderen Pixelnachbarschaften berücksichtigt werden – eine Akkumulatorzelle mit Koordinaten (*r*, 0) ist ein direkter Nachbar der Zelle mit Koordinaten (-*r*, π). Also die gesamte Akkumulatorsäuberung erfolgt folgenderweise, als Pseudocode geschrieben:

```
Wenn kein Benutzerdefinierter Kernel angegeben wurde,
  erzeuge den LoG-Kernel der gewünschten Größe
  und der Standardabweichung;
Vergrößere den Akkumulator um die Hälfte der Breite
  des Kernels nach links und rechts und um die Hälfte
  der Höhe des Kernels nach oben und nach unten und
  setze die Werte der neuen Akkumulatorzellen entsprechend
  der Nachbarschaften;
Wende den linearen Operator an;
Stelle die ursprüngliche Größe des Akkumulator wieder her;
  /* das ist der Akkumulator der ausgegeben wird */
```

Ausgabe der gefundenen Geraden in eine Textdatei

Optional können Parameter der gefundenen Geraden in eine Textdatei abgespeichert werden. Eine Gerade wird als gefunden bezeichnet wenn der Wert der entsprechenden Akkumulatorzelle größer ist als der Schwellwert, der als Eingabeparameter dem Library-Call übergeben wird, und wenn er das lokale Maximum innerhalb des Fensters der Größe des angewandten Kernels ist.

Diese Textdatei enthält Informationen über die Größe des Eingabebildes, über den Koordinatenursprung (bzw. worauf sich die Parameter der gefundenen Geraden beziehen) und über die detektierten Geraden, deren Parameter r und θ in diese Datei abgelegt werden. Im Anhang B wird das Format dieser Datei formal beschrieben.

A.2 Drehtellerkalibrierung

Drehtellerkalibrierung wurde als *kroutine* `TosCalib` implementiert. Es wurde noch eine *kroutine* implementiert, `TosCalibSingle`, die von der Routine `TosCalib` benötigt wird.

A.2.1 kroutine `TosCalib`

Die Eingabe für diese Routine ist das folgende:
- Eines der Eingabebilder mit dem Kegel,
- Das Bild des "leeren" Drehtellers, ohne den Kegel,
- Datei mit den Kameraparametern (fokale Länge und die Pixelgröße in x- und y-Richtung in mm),
- Winkelschrittweite zwischen zwei benachbarten Eingabebildern,
- Höhe des Kegels, in mm.

Die anderen Eingabeparameter dienen zur Ansteuerung der dem Kalibrierverfahren zugrundeliegenden linearen Hough-Transformation. Siehe Anhang B für eine vollständige Beschreibung aller Eingabeparameter.

Die Ausgabe sind drei ASCII-Files:
- Die Transformationsmatrix, die das Bild- in das Objektkoordinatensystem umwandelt,
- Die Parameter der Achsen des Objektkoordinatensystems im Bildkoordinatensystem – dieses File dient nur zur Überprüfung der Ergebnisse – es kann mittels des Tasks `TosLinesToImage` (der ebenfalls im Rahmen dieses Praktikums entwickelt wurde) in ein Binärbild umgewandelt werden, in dem man die Rotationsachse (y-Achse des Objektkoordinatensystems) und die Drehtellerline (x-Achse des Objektkoordinatensystems) sehen kann,
- Ein File mit den Koordinaten der Kegelspitze, sowie mit den Parametern der unteren Seite des Kegels in jedem Eingabebild. Dieses File dient auch zur Überprüfung der Ergebnisse – man kann händisch Überprüfen, ob die Kegelspitze in jedem Eingabebild richtig detektiert wurde.

Die Arbeitsweise der Routine `TosCalib` kann man in folgende Schritte aufteilen:

- Ablesen der Kameraparameter aus dem entsprechenden Eingabefile,
- Bestimmung der Namen der zu bearbeitenden Eingabebilder; es wird angenommen, daß die Namen der Eingabebilder aus 4 Ziffern bestehen (bis auf die Extension), die dem Winkel entsprechen, aus dem das jeweilige Bild aufgenommen wurde. Die durch diese 4 Ziffern dargestellte Zahl, dividiert durch 10, soll dem Aufnahmewinkel in Grad entsprechen. Als erstes Eingabebild wird das als Eingabeparameter angegebene Bild genommen; jedes nächste Bild wird im gleichen Verzeichnis gesucht, mit der gleichen Extension, und sein Name wird durch die 4-Ziffern Codierung des Winkels gebildet, der sich aus der Summe des Winkels des vorhergehenden Bildes und der Winkelschrittweite ergibt,
- Für jedes der Eingabebilder wird ein Kindprozeß erzeugt, der die Routine `TosCalibSingle` aufruft (siehe Abschnitt A.2.2), die das dritte Ausgabefile mit Daten (mit den Koordinaten der Kegelspitze und den Parametern der unteren Seite des Kegels) auffüllt. Das wurde so gemacht, weil bei einer großen Anzahl von Eingabebildern das Kalibrierverfahren sehr viel Speicherplatz benötigt und durch das Erzeugen eines neuen Prozesses für die Bearbeitung des jeden Eingabebildes wird nach dem Beenden des Prozesses wirklich der ganze für die Bearbeitung verwendete Speicherplatz freigegeben.
- Das dritte Ausgabefile besteht aus vier Spalten – x-Koordinate, y-Koordinate der Kegelspitze, Normalabstand r und der Winkel θ der Gerade, die der unteren Kegelseite entspricht; für jede dieser Spalten wird der Mittelwert ermittelt,
- Basierend auf diesen Mittelwerten werden die Parameter der Rotationsachse berechnet, wie es im Abschnitt 4.1 beschrieben wurde,
- Aufgrund der Parameter der Rotationsachse und der Höhe des Kegels wird die Entfernung zwischen der Kamera und der Rotationsachse sowie die Parameter der gesuchten Transformationsmatrix bestimmt, wie es in Abschnitten 4.2 und 4.3 beschrieben wurde.

A.2.2 kroutine `TosCalibSingle`

Als Eingabeparameter nimmt diese Routine ein Bild mit dem Kegel, das leere Drehtellerbild, und die Parameter für die lineare Hough-Transformation. Für das übergebene Eingabebild, macht die Routine `TosCalibSingle` folgendes:

- Das leere Drehtellerbild wird vom Eingabebild abgezogen, damit im Ergebnisbild nahezu nur noch der Kegel übrigbleibt – dadurch werden Störungen zum Großteil eliminiert und die Genauigkeit des Verfahrens erhöht,
- Auf das so entstandene Bild wird die lineare Hough-Transformation angewendet,
- Im von der Hough-Transformation gelieferten Akkumulator werden die Parameter der Geraden gesucht, die den drei Seiten des Kegels entsprechen. Die zwei Schenkel werden durch die Maximumsuche in Bereichen des Akkumulators bestimmt, wo θ im Intervall $[\pi/6, \pi/3]$, bzw. $[2\pi/3, 5\pi/6]$ liegt, und die untere Seite durch die Maximumsuche im θ-Bereich $[4\pi/9, 5\pi/9]$,
- Die Kegelspitze wird als der Schnittpunkt der Schenkel berechnet,
- Die Koordinaten der Kegelspitze, sowie die Parameter der unteren Seite des Kegels werden in das Ausgabefile geschrieben.

Anhang B: Benutzerhandbuch

Im ersten Teil dieses Anhangs wird die Organisation der entwickelten Tasks beschrieben, im zweiten deren Installation und im dritten deren Verwendung. Der dritte Teil ist ein Auszug aus der Manual Pages der Tasks.

B.1 Organisation der Tasks

Alle Tasks sind in zwei Toolboxen zu finden: `tos_imageproc` und `tos_calib`. Die Toolbox `tos_imageproc` beinhaltet die folgenden vier Tasks:

- `TosGradOp` – Gradient-Operator, der neben dem Gradientenbetragsbild auch das Gradientenrichtungsbild als Ergebnis liefert,
- `TosGenLoG` – Generierung eines Laplacian-of-Gaussian Kernels,
- `TosHoughTrans` – die in dieser Arbeit beschriebene lineare Hough-Transformation.
- `TosLinesToImage` – ein Task, der die von der linearen Hough-Transformation ausgegebene Textdatei in ein Binärbild umwandelt.

Für jeden Task gibt es auch den entsprechenden Library-Call, in der ebenfalls in der tos_imageproc Toolbox enthaltenen Library `TosImageProc`.

Die Toolbox `tos_calib` beinhaltet zwei Task:

- `TosCalib` – das in dieser Arbeit vorgestellte Drehtellerkalibrierungsverfahren,
- `TosCalibSingle` – nur intern, von `TosCalib` verwendet.

In *Cantata* befinden sich die Tasks der `tos_imageproc` Toolbox im Untermenü „*TOS Image Processing*“ des „*Glyphs*“ Menüs, wo sie in Kategorien „*Generate Data*“ und „*Image Processing*“ aufgeteilt sind. Der Task der `tos_calib` Toolbox ist im Untermenü „*TOS Calibration*“ des „*Glyphs*“ Menüs zu finden, unter Kategorie „*Turntable calibration*“.

B.2 Installation

Für die Installation der beiden Toolboxen gelten die allgemeinen Regeln für die Installation einer neuen Toolbox auf einen Khoros 2.2 – konformen System:

- Die Distributionsfiles (in diesem Fall `tos_imageproc.tar.gz` und `tos_calib.tar.gz`) in das Verzeichnis kopieren, wo die Toolboxen installiert werden sollen.

- Die Files entpacken („`gunzip <gzip-archive>`“ und „`tar xf <tar-archive>`“).
- Die Referenzen auf die neuen Toolboxen in das `~/.Toolboxes` File (oder `<Khoros-install-dir>/Toolboxes`, falls die Toolboxen systemweit zugreifbar sein sollen):

```
TOS_IMAGEPROC:<Toolbox-install-dir>/tos_imageproc
TOS_CALIB:<Toolbox-install-dir>/tos_calib
```

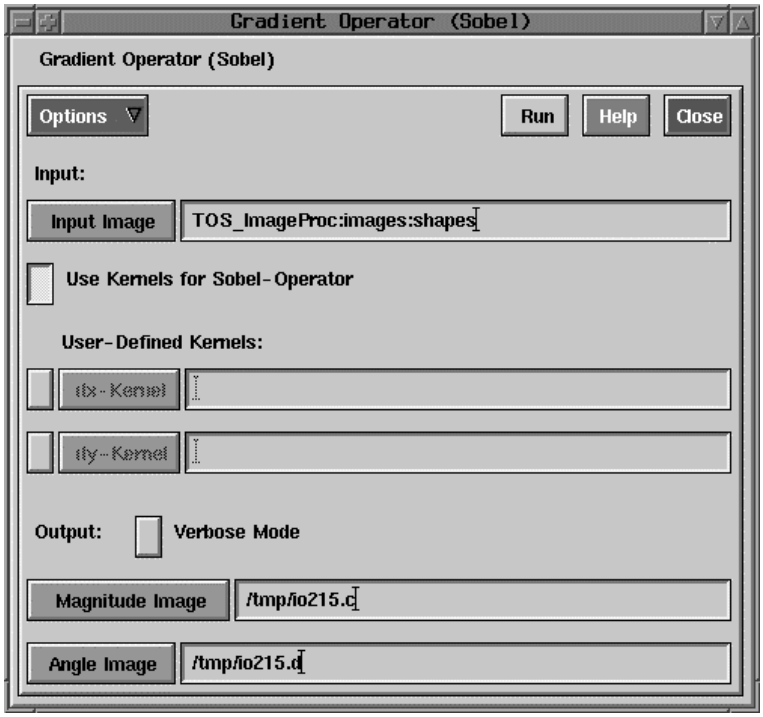
- `Imakefiles` und `Makefiles` der Toolboxen generieren (im `<Toolbox>/objects` Verzeichnis „`make GenImakefile`“, „`make GenImakefiles`“, „`make Makefile`“ und „`make Makefiles`“ ausführen).
- „`make install`“ im `<Toolbox>/objects` Verzeichnis ausführen.

Die `tos_imageproc` Toolbox ist *vor* der `tos_calib` Toolbox zu installieren, da die letzte von der ersten abhängt.

B.3 Verwendung der Tasks

Hier werden die wesentlichen Teile der Manual Pages der Tasks gegeben. Dabei wurde auf die „`DESCRIPTION`“ des Tasks `TosHoughTrans` verzichtet, da dieser Task im Abschnitt 3 und seine Implementierungsdetails im Anhang A ausführlich beschrieben wurden.

B.3.1 `TosGradOp`



PROGRAM

`TosGradOp - Gradient Operator (Sobel)`

DESCRIPTION

TosGradOp performs gradient operation on an image, using
Sobel 3x3 or user defined kernels for x- and y-direction.
Output are both gradient magnitude and gradient angle images.

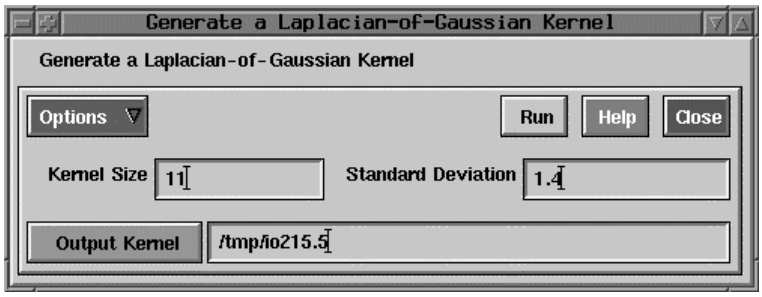
REQUIRED ARGUMENTS

```
-i      type: infile
        desc: input image

-omag   type: outfile
        desc: output image with gradient magnitude information

-oang   type: outfile
        desc: output image with gradient angle information

Mutually Exclusive Group; you must specify ONE of:

-sobel type: flag
        desc: use kernels for Sobel-operator

OR

     ALL OF the Mutually Inclusive Group:

-kernelx
        type: infile
        desc: kernel for x-direction
        default: {none}

     AND

-kernely
        type: infile
        desc: kernel for y-direction
        default: {none}
```

OPTIONAL ARGUMENTS

```
-verbose
        type: flag
        desc: show progress of the task on kstderr
```

B.3.2 TosGenLoG



PROGRAM

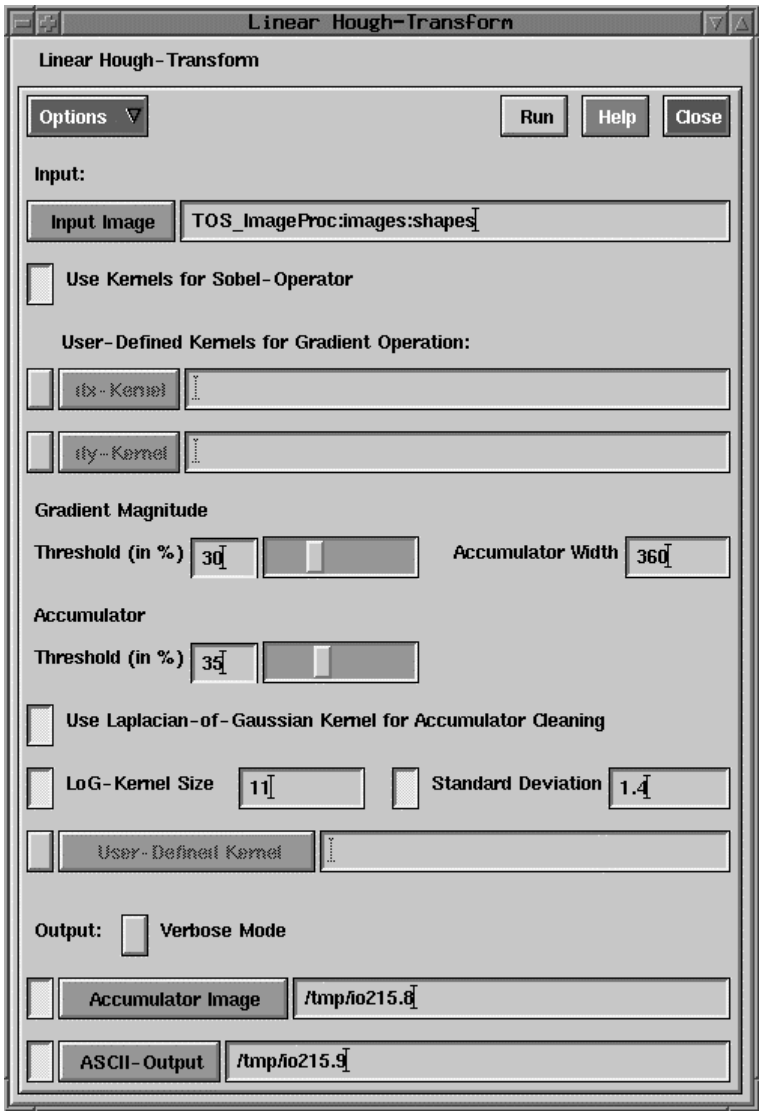
TosGenLoG - Generate a Laplacian-of-Gaussian Kernel

DESCRIPTION

TosGenLoG Generates a Laplacian-of-Gaussian (LoG)-kernel,
which can be used to approximate the second spatial derivative

```
     of an image.  The created kernel can be viewed using the
     putplot3 (Non-Interactive 3D Plot Display) standard
     Khoros task.

REQUIRED ARGUMENTS
     -size  type: integer
            desc: width/height of the output LoG-kernel
            bounds: value > 0

     -sigma type: double
            desc: standard Gaussian deviation of the LoG-kernel
            bounds: value > 0.0

     -o     type: outfile
            desc: output LoG-kernel
```

B.3.3 TosHoughTrans




```
PROGRAM
     TosHoughTrans - Linear Hough-Transform

REQUIRED ARGUMENTS
     -i     type: infile
```

```
       desc: input image

-gthr  type: integer
       desc: threshold value (in percent) in gradient magnitude image
       bounds: 0 < [-gthr] < 100

-wacc  type: integer
       desc: width of the accumulator image
       bounds: value > 0

-athr  type: integer
       desc: threshold value (in percent) in accumulator image
       bounds: 0 < [-athr] < 100

Mutually Exclusive Group; you must specify ONE of:

-sobel type: flag
       desc: use kernels for Sobel-operator

OR

     ALL OF the Mutually Inclusive Group:

-kernelx
       type: infile
       desc: kernel for gradient operator in x-direction
       default: {none}

     AND

-kernely
       type: infile
       desc: kernel for gradient operator in y-direction
       default: {none}

Mutually Exclusive Group; you must specify ONE of:

     ALL OF the Mutually Inclusive Group:

-log   type: flag
       desc: use a LoG-kernel for accumulator cleaning

     AND

-logsize
       type: integer
       desc: size of Laplacian-of-Gaussian Kernel
       default: 11
       bounds: value > 0

     AND

-logsigma
       type: double
       desc: standard Gaussian deviation
       default: 1.3
       bounds: value > 0.0

-kernelacc
       type: infile
       desc: kernel for accumulator cleaning
       default: {none}
```

OPTIONAL ARGUMENTS

-verbose
type: flag
desc: show progess of the task on kstderr

Group; specify AT LEAST ONE of:

-o1 type: outfile
desc: output accumulator image
default: {none}

AND/OR

-o2 type: outfile
desc: output file with parameters of detected lines
default: {none}

ADDITIONAL GUIDELINES FOR USE

- i: input image has to be an 8-bit gray scale image

- gthr: by setting gthr to a high value only significant edges in the input image will be considered relevant for the line detection. This can be desired if there are uninteresting regions in the input image (like for example background) with different gray values and possibly some low visible edges, so if they would be further processed by the Hough-transform, they could lead to unwanted results. A value between 30 and 50 is usually reasonable, but it heavily depends on the input image and which lines should be detected. A low value of gthr favors long lines and a high value favors lines with high edge intensity. If the task "detects" some unexisting lines, this value might have been too low and it should be increased. If almost no lines were detected, even the "expected" ones, this value was probably too high (or the value of athr, see below).

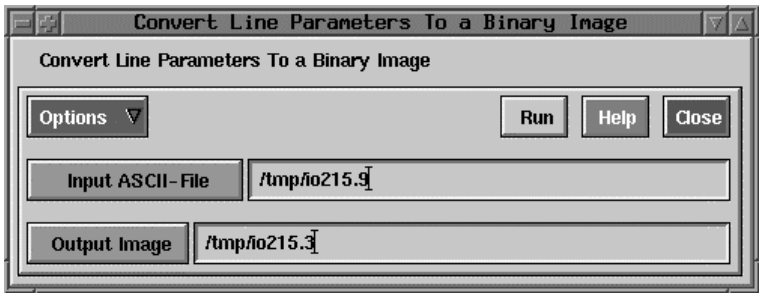
- athr: the value of athr is obsolete if no ASCII output is desired. It has no influence on the output accumulator image, but only on the number of lines whose (RAD, THETA)-parameters that will be written to the ASCII output file. The higher the number of lines is that should be detected, the smaller the value of this parameter should be. Also, if the size of the used LoG-kernel (logsize) is increased, the value of athr should be decreased, because by using a large LoG-kernel it is more likely to happen that there will be a pixel in the accumulator with an extremely high value comparing to all the other accumulator pixels.

- wacc: the value of wacc tells directely, lines with how many different angles can be represented by the accumulator. If wacc is, for example, 90, then the angle of all detected lines will be an even number (of degrees), because angles are in the range between 0 and 180 degrees, and wacc of 90 means that 90 different angle values between 0 and 180 degrees can be represented by the accumulator. Values greater than 180 are rarely necessary.

- sobel, kernelx, kernely: use of -sobel flag is recommended, because using larger kernels doesn't really improve the results of Hough-transform, unless there is a lot of noise in the input image.

- log, logsize, logsigma, kernelacc: use of a LoG-kernel is suitable for accumulator cleaning. Which size (log-size) and standard deviation (logsigma) is appropriate, depends on the input image. For images without noise and with almost straight lines, a small kernel (size 5x5 or less) is sufficient, even using no kernel at all (by "using" a kernel with size 1x1) could give good results. But for the most of the real-world images, use of 9x9 or larger kernels is recommended. Standard deviation (logsigma) should be big enough for a given logsize, so that it really makes sense to use a kernel of that size. A general guideline: logsigma = 0.14 * logsize.

B.3.4 TosLinesToImage



PROGRAM

TosLinesToImage - Convert Line Parameters To a Binary Image

DESCRIPTION

TosLinesToImage creates a binary image based on information written in an ASCII-file. This task was developed as support for the task TosHoughTrans, which detects lines within an image and writes them in their (RAD, THETA) parametric form in an ASCII-file. For more information on (RAD, THETA) parametric form, see TosHoughTrans(1) manual page.

DESCRIPTION OF THE INPUT FILE FORMAT

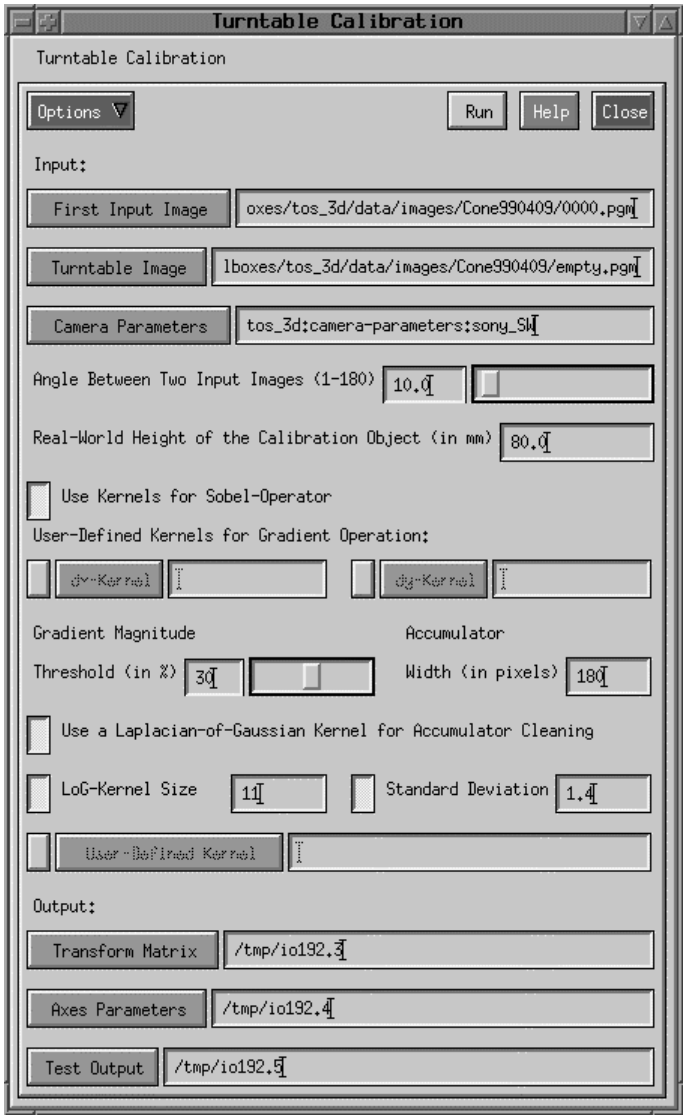
```
ImageSize <width-of-image> <height-of-image>
ImageHotspot <x0> <y0>
Detected Lines
   <rad0>  <theta0>
   <rad1>  <theta1>
   <rad2>  <theta2>
   ...
```

The first section (the ImageSize keyword followed by width and height of the output image) specifies the size of the output image, the second section (the ImageHotspot keyword followed by 2 values) specifies the (x, y)-coordinates of the origin of the output image coordinate system and the third section (the DetectedLines keyword) says that what follows, represents the detected lines, in their (RAD, THETA) parametric form. The order of the sections is not important. Comments are allowed anywhere in the file, they just must have '#' as the first character of the line. The x-axis of the image coordinate system is taken to be directed from left to right and the y-axis from up to down.

REQUIRED ARGUMENTS

```
     -i      type: infile
             desc: input ASCII-file

     -o      type: outfile
             desc: output binary image
```

B.3.5 `TosCalib`



PROGRAM

TosCalib - Turntable Calibration

DESCRIPTION

TosCalib is a task which can be used for calibration of a computer vision system that basicaly consists of a camera and turntable, if the optical axis of the camera lies approximetaly in the turntable plane, orthogonal to the rotation axis of the turntable. The task computes the transform matrix for converting the image coordinate system, with origin in the center of the image and x-axis directed from left to right and y-axis from up to down, into the

object coordinate system, with the origin in the intersecting point of the turntable plane and it's rotation axes, with x-axis parallel to the x-axis of the image coordinate system, y-axis going upwards from the origin and the z-axis directed to the camera. All the points in the (2D) image coordinate system will be mapped to a (3D) point in the object coordinate system with z-coordinate equal 0, after being multiplied with the output transform matrix.

A cone-like object must be used for calibration. The calibration is based on detecting the peak of the cone in different input images, created by rotating the turntable for a certain angle.

The 3 required input files are:

- the first of the input images, also indicating in which directory the input images can be found. All the input files must have a 4-digit filename indicating the angle from which the image was taken, in tenths of a degree,

- an image of the turntable without the cone on it,

- a file with camera parameters. This file will be looked up for the lines:
  f=<focal_length>
  dx=<pixel_size_in_x_direction>
  dy=<pixel_size_in_y_direction>

The other input parameters are the angle between two neighbouring input images and the real-world size of the cone. The rest of the parameters is passed to the linear HoughTransform, which is the base of this calibration algorithm. See TosHoughTrans(1) for the exact description of these parameters.

The first output file is the transform matrix, the second an ASCII file containing parameters of the axes of the object coordinate system, which can be converted to a binary image using TosLinesToImage(1) task, and the third output file is an ASCII file containing the detected cone peaks (their x- and y-coordinate) and (RAD, THETA)-parameters of the detected turntable line in each of the processed input images. This file can be used for manual testing of the calibration task. The task produces also some information about its progress on the kstdout.

REQUIRED ARGUMENTS

-i1 type: infile
desc: first input image in the directory

-i2 type: infile
desc: turntable image without a cone

-i3 type: infile
desc: input file with camera parameters

-step type: double
desc: angle between two input images, in degrees (1-180)
bounds: 1 < [-step] < 180

-h type: double

```
       desc: real-world height (in mm) of the calibration
       object (cone)
       bounds: value > 0.0

-gthr  type: integer
       desc: threshold value (in percent) in gradient magni-
       tude image
       bounds: 0 < [-gthr] < 100

-wacc  type: integer
       desc: width (in pixels) of the accumulator image
       bounds: value > 0

-o1    type: outfile
       desc: transform matrix image->object coordinate sys-
       tem

-o2    type: outfile
       desc: output file with rotation and turntable axes
       parameters

-o3    type: outfile
       desc: output file with cone peaks and line parameters

Mutually Exclusive Group; you must specify ONE of:

-sobel type: flag
       desc: use kernels for Sobel-operator

OR

     ALL OF the Mutually Inclusive Group:

-kernelx
       type: infile
       desc: kernel for gradient operator in x-direction
       default: {none}

     AND

-kernely
       type: infile
       desc: kernel for gradient operator in y-direction
       default: {none}

Mutually Exclusive Group; you must specify ONE of:

     ALL OF the Mutually Inclusive Group:

-log   type: flag
       desc: use a LoG-kernel for accumulator cleaning

     AND

-logsize
       type: integer
       desc: size of the Laplacian-of-Gaussian Kernel
       default: 5
       bounds: value > 0

     AND

-logsigma
```

```
        type: double
        desc: standard Gaussian deviation of the LoG-Kernel
        default: 1
        bounds: value >= 0.0

-kernelacc
        type: infile
        desc: kernel for accumulator cleaning
        default: {none}
```