Detektion von Eckpunkten und einfachen Kurven

Industrielle Bildverarbeitung, Vorlesung No. 7[1]

M. O. Franz

21.11.2007

[1] falls nicht anders vermerkt, sind die Abbildungen entnommen aus Burger & Burge, 2005.

Übersicht

Übersicht

Eckpunkte

Eckpunkte in Bildern sind die Basis für eine Vielzahl von Anwendungen:

- Verfolgung von Objekten in aufeinanderfolgenden Videobildern (tracking)
- Zuordnung von Bildstrukturen in Stereoaufnahmen
- Referenzpunkte zur geometrischen Vermessung mit einem oder vielen Bildern
- Kalibrierung von Kameras
- als Ankerpunkte bei der Segmentierung von Objekten in seine Teile

Eckpunkte sind robuste Merkmale: sie bleiben in einem breiten Bereich von Ansichtswinkeln und Beleuchtungsbedingungen detektierbar.

Detektion von Eckpunkten

Kanten: Bildbereiche, in denen der Gradient in *einer* Richtung hoch und senkrecht dazu niedrig ist.

Eckpunkte: Bildbereiche, in denen der Gradient in mehr als einer Richtung hoch ist.

Gewünschte Eigenschaften:

- Unterscheidung von wichtigen und unwichtigen Eckpunkten
- Zuverlässiges Auffinden von Eckpunkten unter Bildrauschen
- Genaue Lokalisierung der Eckpunkte
- Möglichst wenig Rechenaufwand
- Unabhängig von der Orientierung der Ecken

Harris-Detektor (1)

Partielle Bildableitung in horizontaler und vertikaler Richtung:

$$\partial_x I(u,v) = \frac{\partial I}{\partial x}(u,v) \quad \text{und} \quad \partial_y I(u,v) = \frac{\partial I}{\partial y}(u,v)$$

Daraus Berechnung der lokalen Strukturmatrix:

$$M = \begin{pmatrix} \partial_x I^2 & \partial_x I \partial_y I \\ \partial_x I \partial_y I & \partial_y I^2 \end{pmatrix}$$

Gewichtete Mittelung von M mit Gaußfilter H_σ:

$$M = \begin{pmatrix} \partial_x I^2 * H_\sigma & \partial_x I \partial_y I * H_\sigma \\ \partial_x I \partial_y I * H_\sigma & \partial_y I^2 * H_\sigma \end{pmatrix} = \begin{pmatrix} A & C \\ C & B \end{pmatrix}$$

Eigenwerte:

$$\lambda_{1,2} = \frac{tr(M)}{2} \pm \sqrt{\frac{tr(M)^2}{4} - \det M} = \frac{1}{2}(A + B \pm \sqrt{A^2 - 2AB + B^2 + 4C^2})$$

Harris-Detektor (2)

Interpretation der Eigenwerte (beide sind positiv): Eigenwerte codieren die Kantenstärke, Eigenvektoren die Kantenrichtung.

1. In uniformen Bildregionen ist M nahe an 0 and damit auch λ_1 und $\lambda_2 \Rightarrow \lambda_1 = 0, \lambda_2 = 0$.
2. An Kanten ist der Gradient nur senkrecht zur Sprungkante größer als 0, entlang der Kante ist er 0 $\Rightarrow \lambda_1 > 0, \lambda_2 = 0$.
3. An Eckpunkten ist der Gradient in mehr als einer Richtung größer als 0 $\Rightarrow \lambda_1 > 0, \lambda_2 > 0$.

Differenz der Eigenwerte (möglichst klein):

$$\lambda_1 - \lambda_2 = 2\sqrt{\frac{tr(M)^2}{4} - \det M}$$

"Eckenstärke" mit Empfindlichkeitsparameter $\alpha \in [0.04..0.06]$:

$$Q(u,v) = \det M - \alpha tr(M)^2 = (AB - C^2) - \alpha(A+B)^2$$

Ecken werden detektiert, wenn $Q(u,v)$ einen Schwellwert überschreitet.

Harris-Detektor: Algorithmus (1)

1: HarrisCorners($I(u,v)$)
2: Prefilter (smooth) the original image: $I' \leftarrow I * H_p$
3: Step 1 – compute the corner response function:
4: Compute the horizontal and vertical derivatives:
$I_x \leftarrow I' * H_{dx}, \quad I_y \leftarrow I' * H_{dy}$
5: Compute the components of the local structure matrix
$M = \left(\begin{smallmatrix} A & C \\ C & B \end{smallmatrix}\right)$: $A \leftarrow I_x^2, \quad B \leftarrow I_y^2, \quad C \leftarrow I_x I_y$
6: Blur each components of the structure matrix: $\bar{M} = \left(\begin{smallmatrix} \bar{A} & \bar{C} \\ \bar{C} & \bar{B} \end{smallmatrix}\right)$:
$\bar{A} \leftarrow A * H_b, \quad \bar{B} \leftarrow B * H_b, \quad \bar{C} \leftarrow C * H_b$
7: Compute the corner response function:
$Q \leftarrow (\bar{A} \cdot \bar{B} - \bar{C}^2) - \alpha \cdot (\bar{A} + \bar{B})^2$
8: Step 2 – collect corner points:
9: Create an empty list $Corners \leftarrow \{\}$
10: **for** all image coordinates (u,v) **do**
11: **if** $Q(u,v) > t_H$ **and** IsLocalMax(Q,u,v) **then**
12: Create corner node $\boldsymbol{c}_i = (u_i, v_i, q_i) \leftarrow (u, v, Q(u,v))$
13: Add $\boldsymbol{c}_i$ to $Corners$
14: Sort $Corners$ by q_i in descending order.
15: $GoodCorners \leftarrow$ CleanUpNeighbors$(Corners)$
16: **return** $GoodCorners$.

Harris-Detektor: Algorithmus (2)

17: IsLocalMax(Q, u, v) ▷ determine if $Q(u, v)$ is a local maximum
18: Let $q_c \leftarrow Q(u, v)$ (center pixel)
19: Let $\mathcal{N} \leftarrow Neighbors(Q, u, v)$ ▷ values of all neighboring pixels
20: **if** $q_c > q_i$ for all $q_i \in \mathcal{N}$ **then**
21: **return** *true*
22: **else**
23: **return** *false*.

24: CleanUpNeighbors$(Corners)$ ▷ *Corners* is sorted by descending q
25: Create an empty list $GoodCorners \leftarrow \{\}$
26: **while** *Corners* is not empty **do**
27: $\boldsymbol{c}_i = (u_i, v_i, q_i) \leftarrow$ RemoveFirst$(Corners)$
28: Add $\boldsymbol{c}_i$ to *GoodCorners*
29: Delete all nodes $\boldsymbol{c}_j$ from *Corners* if $Dist(\boldsymbol{c}_i, \boldsymbol{c}_j) < d_{\min}$
30: **return** *GoodCorners*.

Harris-Detektor: Parameterwerte

Pre-Filter (Zeile 2): Vorglättung mit einem kleinen, xy-separierbaren Filter $H_p = H_{px} * H_{py}$, wobei

$$H_{px} = \frac{1}{9}\begin{bmatrix}2 & \mathbf{5} & 2\end{bmatrix} \quad \text{und} \quad H_{py} = H_{px}^T = \frac{1}{9}\begin{bmatrix}2\\ \mathbf{5}\\ 2\end{bmatrix}$$

Gradientenfilter (Zeile 4): Berechnung der partiellen ersten Ableitungen in x- und y-Richtung mit

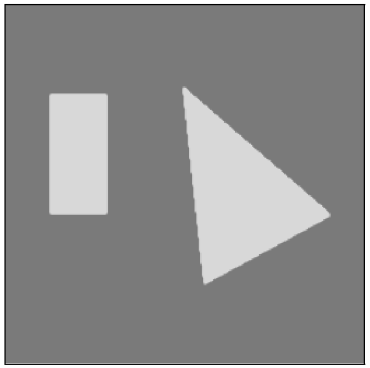
$$H_{dx} = \begin{bmatrix}-0.453014 & \mathbf{0} & 0.453014\end{bmatrix} \quad \text{und} \quad H_{dy} = H_{dx}^T = \begin{bmatrix}-0.453014\\ \mathbf{0}\\ 0.453014\end{bmatrix}$$

Blur-Filter (Zeile 6): Glättung der einzelnen Komponenten der Strukturmatrix M mit separierbaren Gauß-Filtern $H_b = H_{bx} * H_{by}$, mit

$$H_{bx} = \frac{1}{64}\begin{bmatrix}1 & 6 & 15 & \mathbf{20} & 15 & 6 & 1\end{bmatrix} \quad \text{und} \quad H_{by} = H_{bx}^T = \frac{1}{64}\begin{bmatrix}1\\ 6\\ 15\\ \mathbf{20}\\ 15\\ 6\\ 1\end{bmatrix}$$

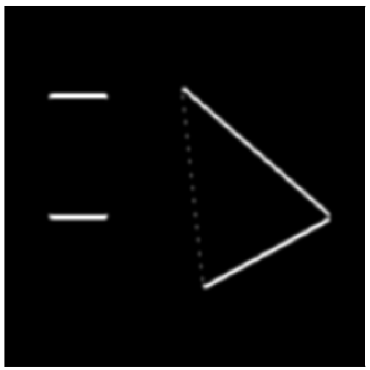
Steuerparameter (Zeile 7): $\alpha = 0.04 \ldots 0.06$ (default 0.05)
Response-Schwellwert (Zeile 13): $t_H = 10.000 \ldots 1.000.000$ (default 25.000)
Umgebungsradius (Zeile 29): $d_{\min} = 10$ pixels

Harris-Detektor: Beispiel

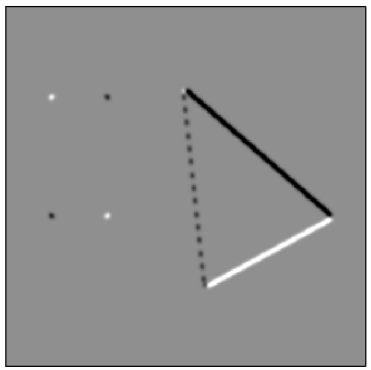


$I(u, v)$

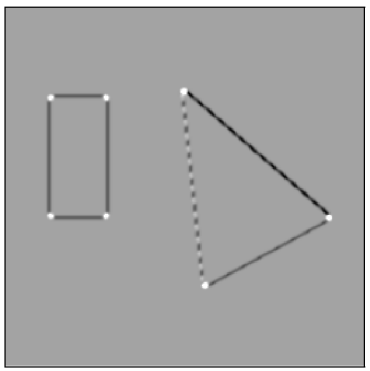
$A = I_x^2(u, v)$



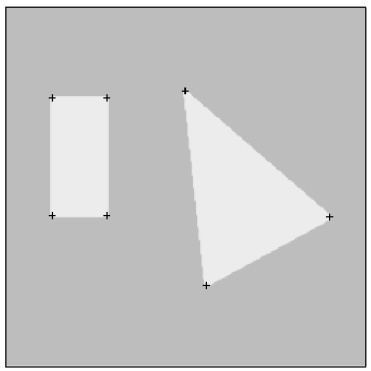
$B = I_y^2(u, v)$



$C = I_x I_y(u, v)$

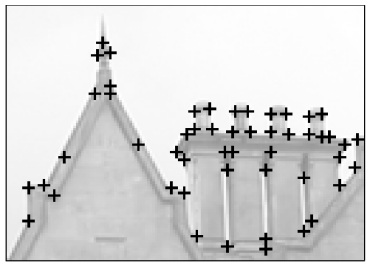


$Q(u, v)$

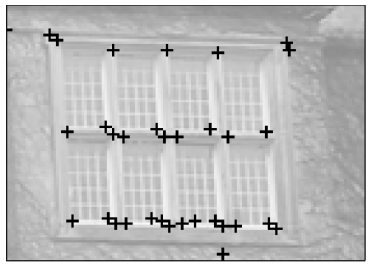


Eckpunkte

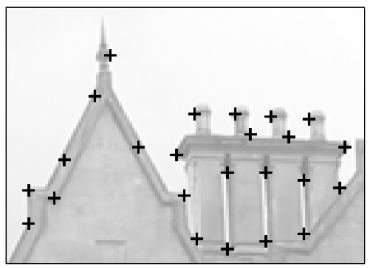
(a)



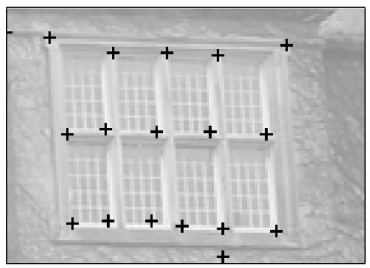
(b)



(c)



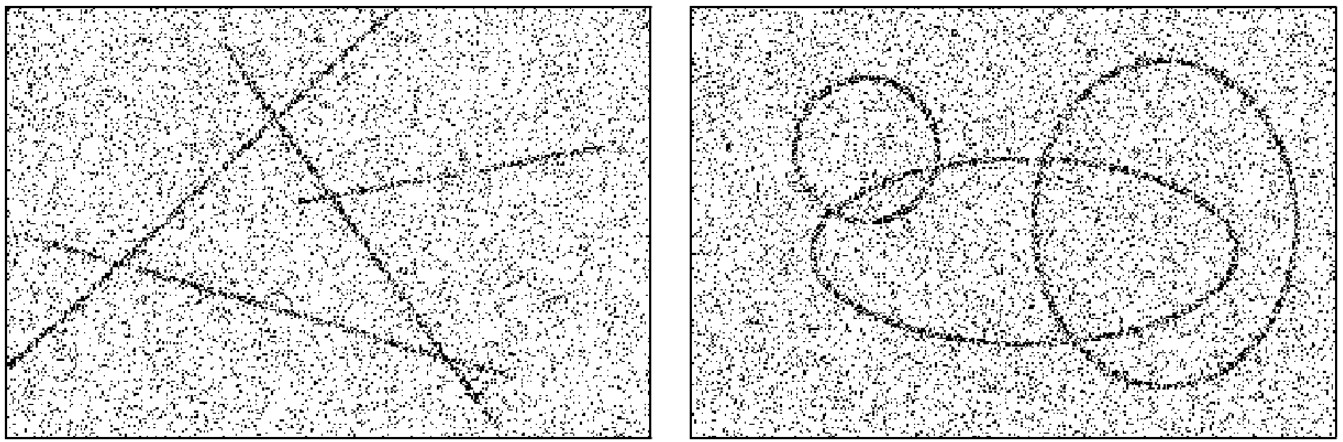
(d)



(e)

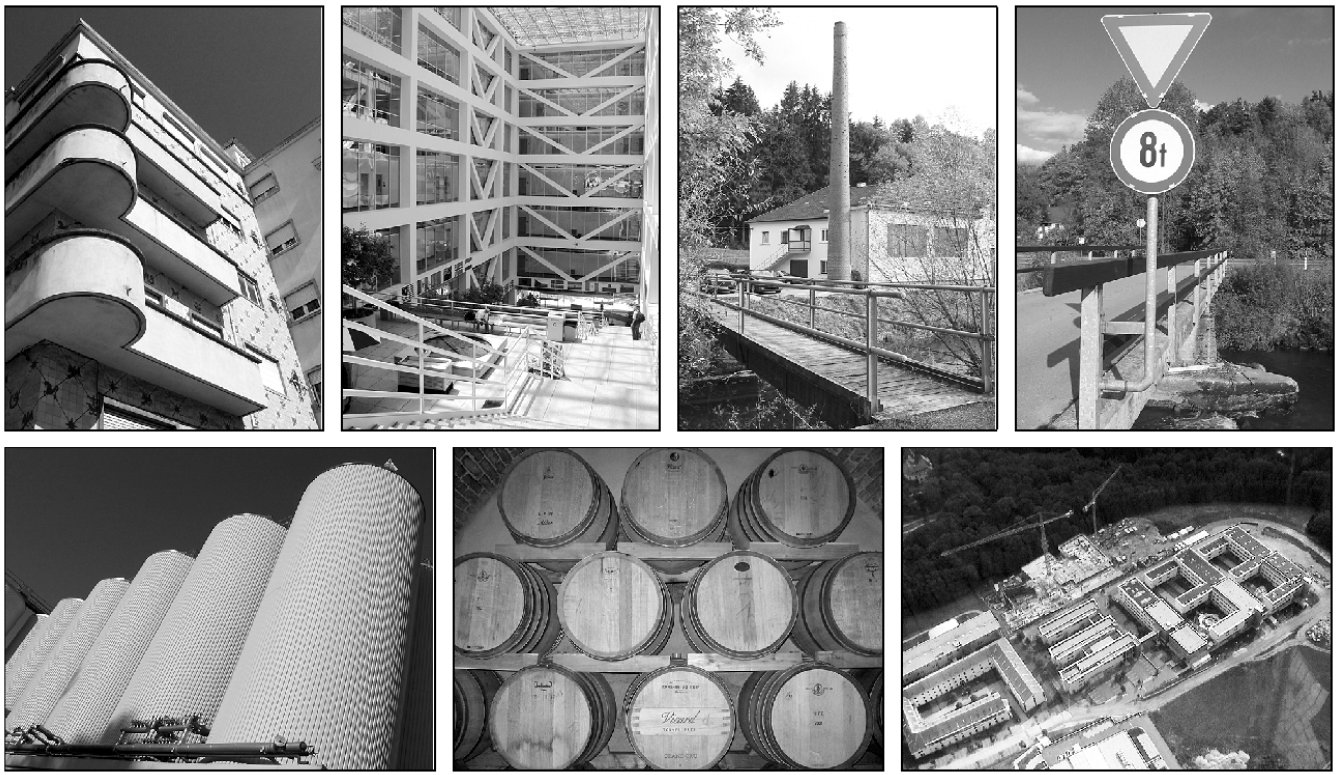
Übersicht

Kantenverfolgung



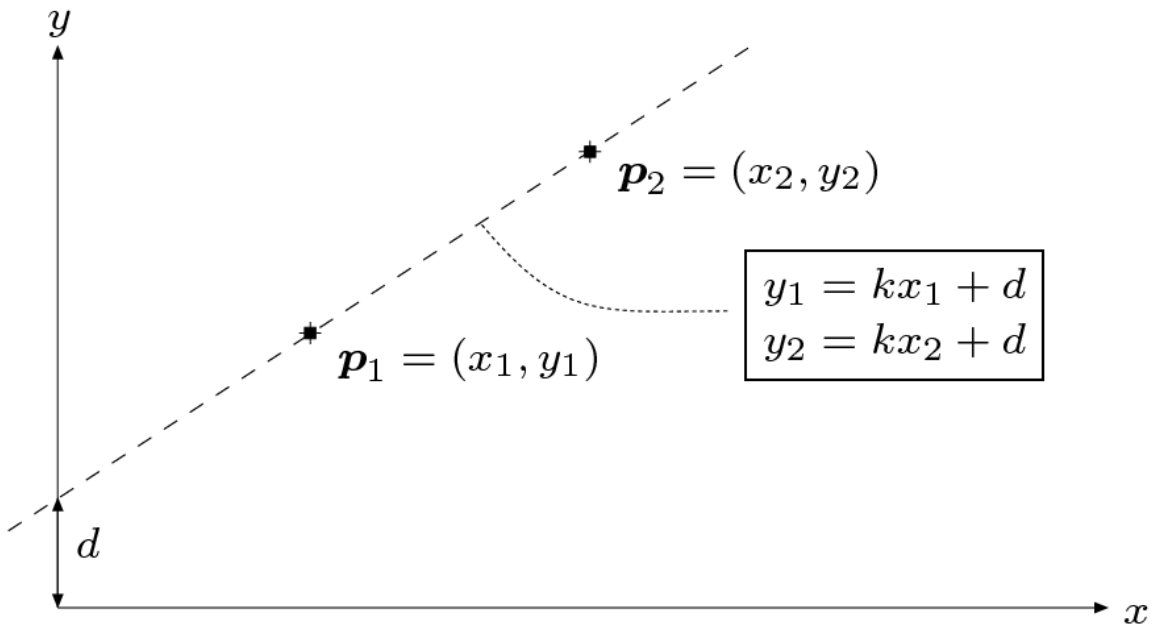
- Kantendetektoren produzieren eine Vielzahl von irrelevanten Kanten, zusätzlich sind die wichtigen Kanten oft unzusammenhängend.
- Kantenverfolgung ist daher ein schwieriges, noch nicht gelöstes Problem (Verzweigungen, Verschmelzung von Kanten usw.).
- Hier: Suche nach einfachen geometrischen Konturen, die sich durch parametrisierte Formeln beschreiben lassen.

Hough-Transformation



Mit der Hough-Transformation lassen sich beliebige, parametrisierbare Formen in Punktverteilungen lokalisieren (z.B. Geraden, Kreise, Ellipsen). Sie ist daher besonders geeignet zur Detektion künstlicher Objekte.

Beispiel für eine parametrisierbare Form: Gerade



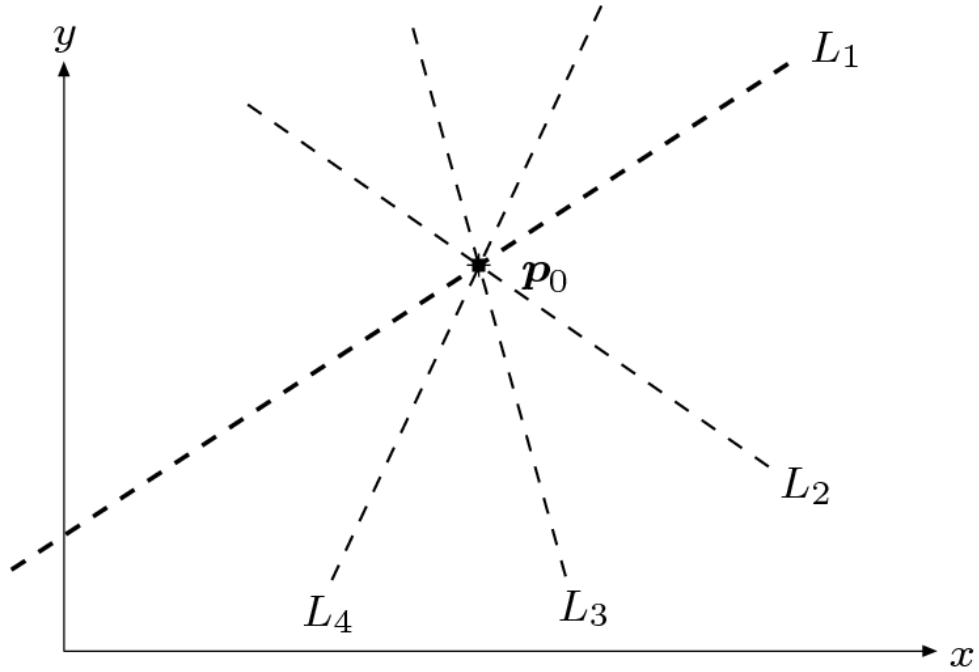
Zweidimensionale Geradengleichung:

$$y = kx + d$$

2 Parameter: Steigung k und y-Achsenabschnitt d. Für eine Gerade, die durch 2 Punkte $\mathbf{p}_1 = (x_1, y_1)$ und $\mathbf{p}_2 = (x_2, y_2)$ gilt

$$y_1 = kx_1 + d \quad \text{und} \quad y_2 = kx_2 + d$$

Parameterraum



Ziel: Auffinden der Geraden mit Parametern k und d, auf denen möglichst viele Punkte liegen.

Die Hough-Transformation sucht im von k und d gebildeten zweidimensionalen Parameterraum alle Geraden, die durch einen gegebenen Punkt $\mathbf{p}_0 = (x_0, y_0)$ laufen.

Geraden im Bild- und Parameterraum (1)

Beliebige Gerade L_j durch $\mathbf{p}_0$:

$$L_j \; : \; y_0 = k_j x_0 + d_j$$

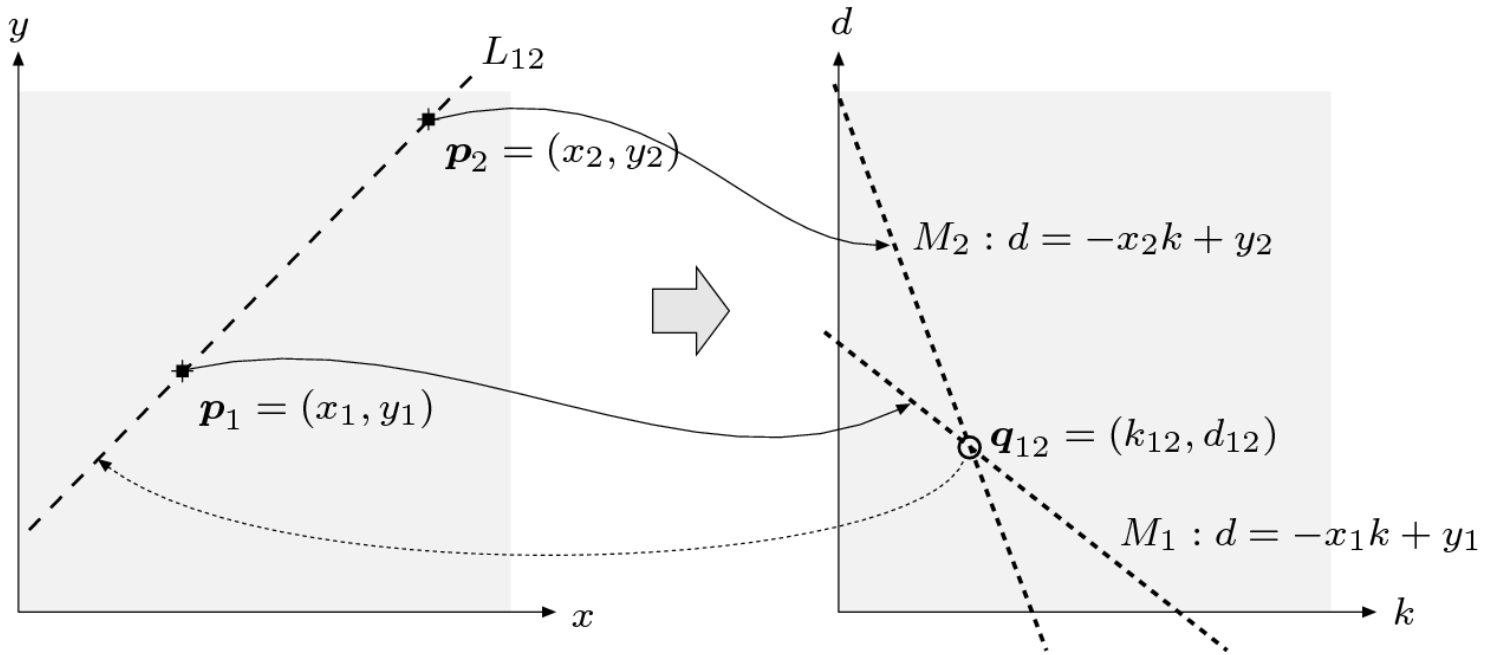
Im Parameterraum ist die Menge aller Geraden durch $\mathbf{p}_0$ ebenfalls eine Gerade:

$$d_j = -x_0 k_j + y_0.$$

Für beliebige Punkte gilt also folgende Beziehung:

Bildraum (x, y)		*Parameterraum* (k, d)	
Punkt	$\boldsymbol{p}_i = (x_i, y_i)$	$M_i : d = -x_i k + y_i$	Gerade
Gerade	$L_j : y = k_j x + d_j$	$\boldsymbol{q}_j = (k_j, d_j)$	Punkt

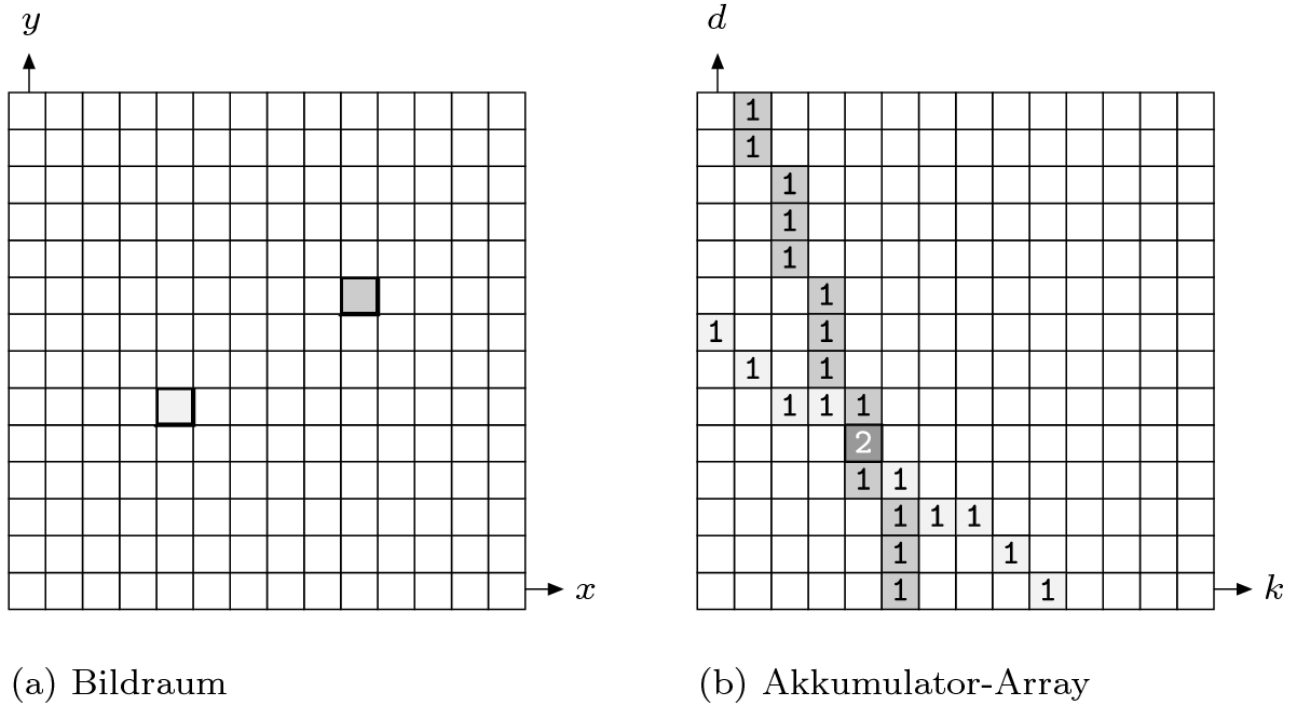
Geraden im Bild- und Parameterraum (2)



(a) x/y-Bildraum (b) k/d-Parameterraum

Wenn sich n Geraden im Parameterraum an Position (k', d') schneiden, dann liegen auf der entsprechenden Geraden $y = k'x + d'$ im Bildraum insgesamt n Bildpunkte.

Akkumulator-Array

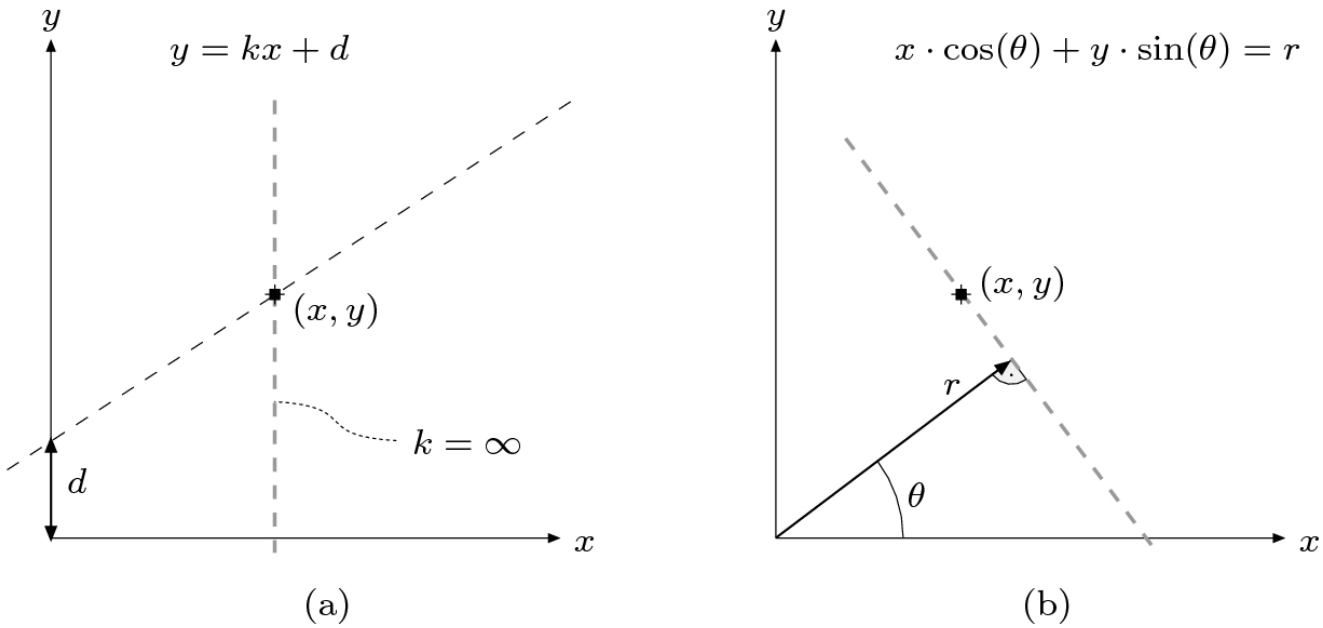


(a) Bildraum (b) Akkumulator-Array

Akkumulator-Array: Diskrete Repräsentation des Parameterraumes.

Grundidee der Hough-Transformation: Für jeden gefundenen Bildpunkt $\mathbf{p}_0$ werden die Zähler im Akkumulator-Array entlang der Geraden $d_j = -x_0 k_j + y_0$ um 1 erhöht.

Eine bessere Geradenparametrisierung



Problem: Vertikale Geraden haben Steigung $k = \infty$.

Hessesche Normalform: $x \cos\theta + y \sin\theta = r$

mit $0 \leq \theta < \pi$ und $-r_{\text{max}} \leq r \leq r_{\text{max}}$ mit $r_{\text{max}} = \frac{1}{2}\sqrt{M^2 + N^2}$.

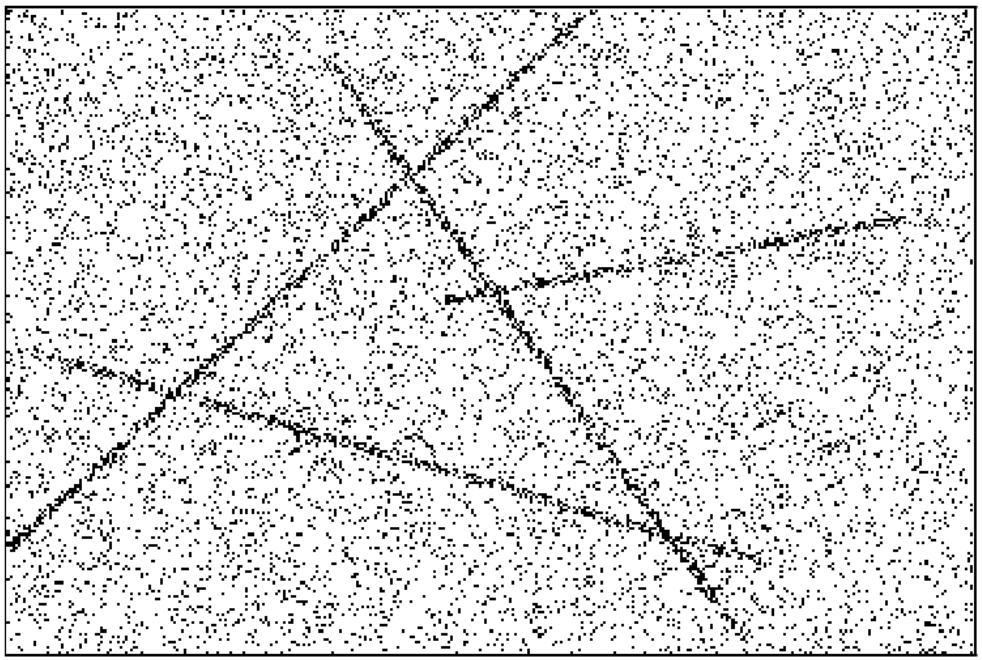
Hough-Algorithmus für Geraden

```
 1: HoughLines(I)
 2:     Set up a two-dimensional array Acc[θ, r] of counters, initialize to 0
 3:     Let (u_c, v_c) be the center coordinates of the image I
 4:     for all image coordinates (u, v) do
 5:         if I(u, v) is an edge point then
 6:             (x, y) ← (u − u_c, v − v_c)        ▷ relative coordinate to center
 7:             for θ_i = 0 ... π do
 8:                 r_i = x cos(θ_i) + y sin(θ_i)
 9:                 Increment Acc[θ_i, r_i]
10:     MaxLines ← FindMaxLines(Acc, K)
11:         ▷ return the list of parameter pairs (θ_j, r_j) for K strongest lines
12:     return MaxLines.
```

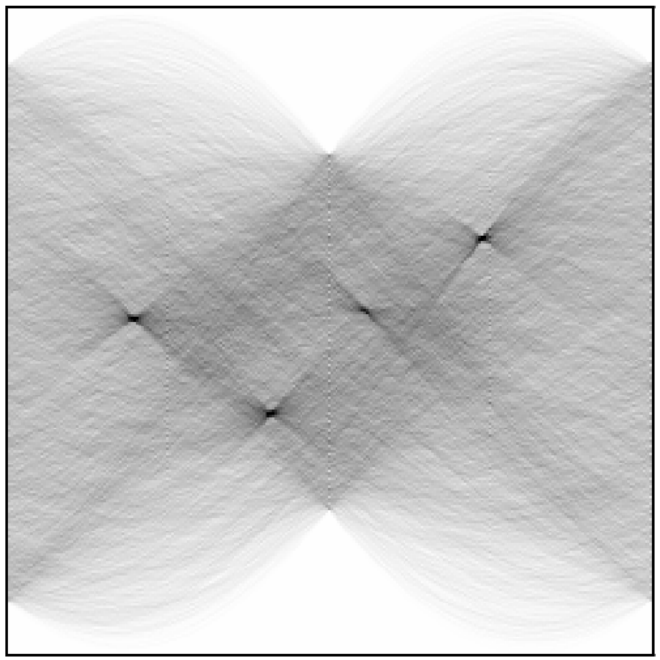
Programmbeispiel

```
9   LinearHT(ImageProcessor ip, int aSteps, int rSteps) {
10    this.ip = ip;
11    xCtr = ip.getWidth()/2; yCtr = ip.getHeight()/2;
12    nAng = aSteps; dAng = (Math.PI/nAng);
13    nRad = rSteps;
14    double rMax = Math.sqrt(xCtr*xCtr + yCtr*yCtr);
15    dRad = (2*rMax)/nRad;
16    houghArray = new int[nAng][nRad];
17    fillHoughAccumulator();
18  }
19
20  void fillHoughAccumulator() {
21    for (int v = 0; v < ip.getHeight(); v++) {
22      for (int u = 0; u < ip.getWidth(); u++) {
23        if (ip.getPixel(u, v) > 0) {
24          doPixel(u, v);
25        }
26      }
27    }
28  }
29
30  void doPixel(int u, int v) {
31    int x = u-xCtr, y = v-yCtr;
32    for (int a = 0; a < nAng; a++) {
33      double theta = dAng * a;
34      int r = (int) Math.round(
35        (x*Math.cos(theta) + y*Math.sin(theta)) / dRad) + nRad
              /2;
36      if (r >= 0 && r < nRad) {
37        houghArray[a][r]++;
38      }
39    }
40  }
41 }
```

Beispiel Hough-Transformation in Hesse-Parameterraum



(a)



(b)

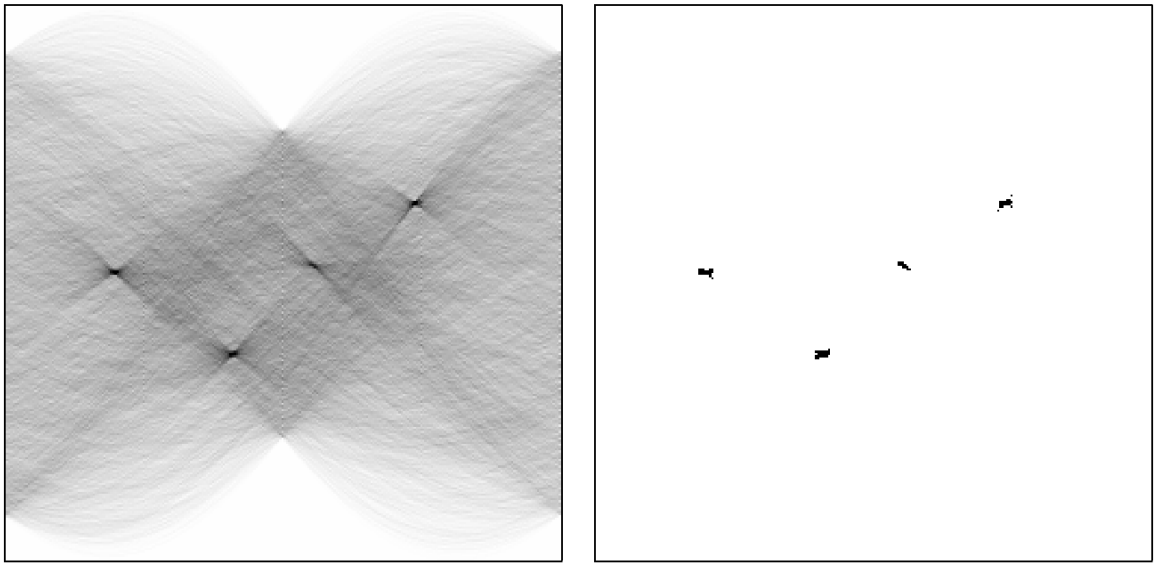
Auswertung des Akkumulator-Arrays

Problem: Die Sinuskurven schneiden sich nicht genau an einem Punkt, sondern in einer Region. Die Lokalisierung der Maxima ist daher der schwierigste Teil der Hough-Transformation.

Ansatz A: Schwellwerte. Alle Akkumulatorzellen unterhalb eines Schwellwertes werden verworfen. Die übrigen werden mit einer morphologischen *Closing*-Operation bereinigt (s. nächste Vorlesung) und anschließend der Schwerpunkt der Regionen bestimmt (s. übernächste Vorlesung).

Ansatz B: Non-Maximum-Supression. Alle Nicht-Maxima werden verworfen, d.h. alle Zellen, deren Einträge nicht größer als die aller Nachbarn sind. Anschließend werden die größten Werte mit einer Schwellwertoperation gefunden.

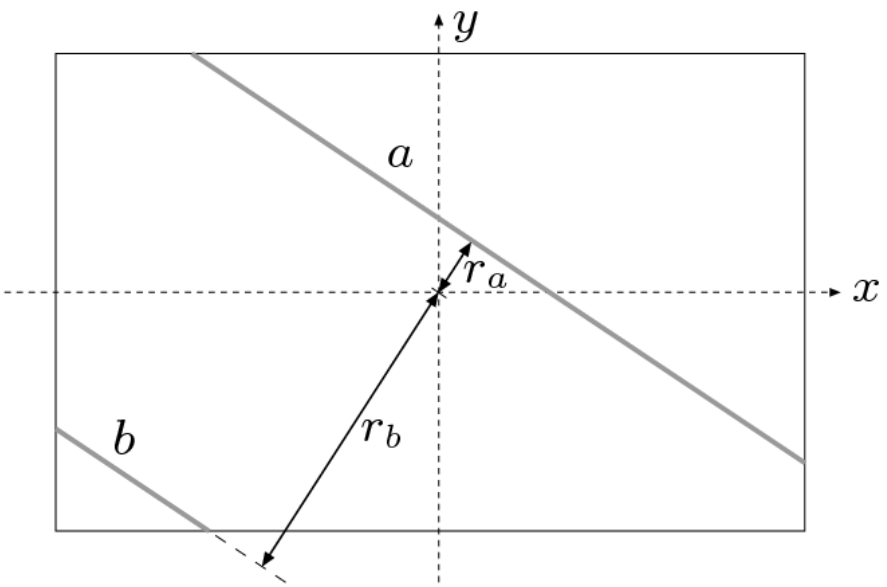
Beispiel: Auswertung des Akkumulator-Arrays



(a) (b)

(c) (d)

Bias-Problem



Problem: Gewicht einer Geraden bestimmt sich aus ihrer Länge, aber weit vom Bildzentrum hat es oft zuwenig Platz für lange Geraden $\Rightarrow$ bestimmte Teile des Akkumulator-Arrays haben nicht die gleiche Füllwahrscheinlichkeit wie andere (Bias).

Ansatz: Normierung mit der Anzahl $n_{\max}[\theta, r]$ der überhaupt möglichen Geraden

$$Acc'[\theta, r] = \frac{Acc[\theta, r]}{n_{\max}[\theta, r]},$$

Bestimmung von $n_{\max}[\theta, r]$ über vollständig oder zufällig gefülltes Bild.

Erweiterungen der Hough-Transformation

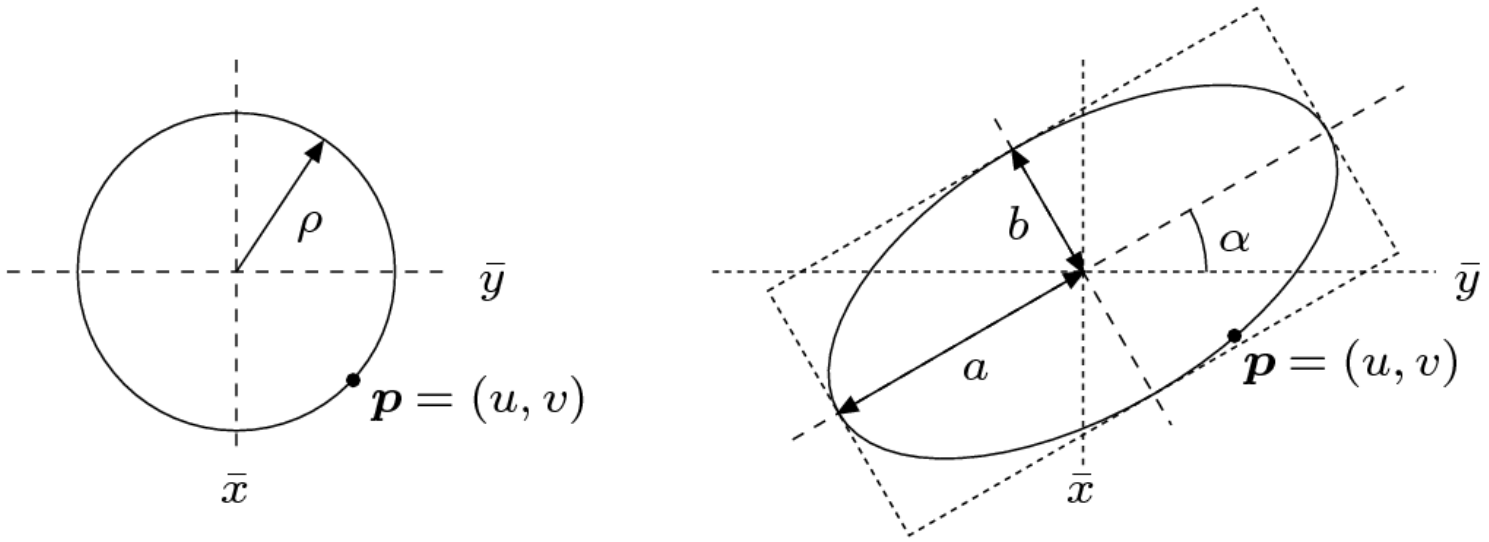
- Endpunkte von Bildgeraden. Das nachträgliche Aufsuchen von Endpunkten ist aufwendig und wenig robust. Beim Füllen des Akkumulator-Arrays kann man hierzu die jeweils maximalen bzw. minimalen x/y-Koordinaten der Punkte mitspeichern, d.h.

$$Acc[\theta, r] = (\text{count}, \text{start}_x, \text{start}_y, \text{end}_x, \text{end}_y)$$

- Berücksichtigung von Kantenstärke und -orientierung. Statt den Akkumulator um 1 zu erhöhen, kann stattdessen die Kantenstärke aufaddiert werden. Wenn die Orientierung bekannt ist, müssen nur die damit kompatiblen Zellen mit der entsprechenden Winkelkoordinate hochgezählt werden.
- Hierachische Hough-Transformation. Zuerst Suche in grob gerastertem Parameterraum, dann feinere Abtastung um die Maxima herum.

Übersicht

Parametrisierung von Kreisen und Ellipsen



Kreise hängen nicht nur von 2, sondern von 3 Parametern ab: x- und y-Position $(\bar{x}, \bar{y})$ des Mittelpunkts und Radius ρ:

$$(u - \bar{x})^2 + (v - \bar{y})^2 = \rho^2$$

Wir benötigen daher ein dreidimensionales Akkumulator-Array, um Kreise (und Kreisbögen) zu finden.

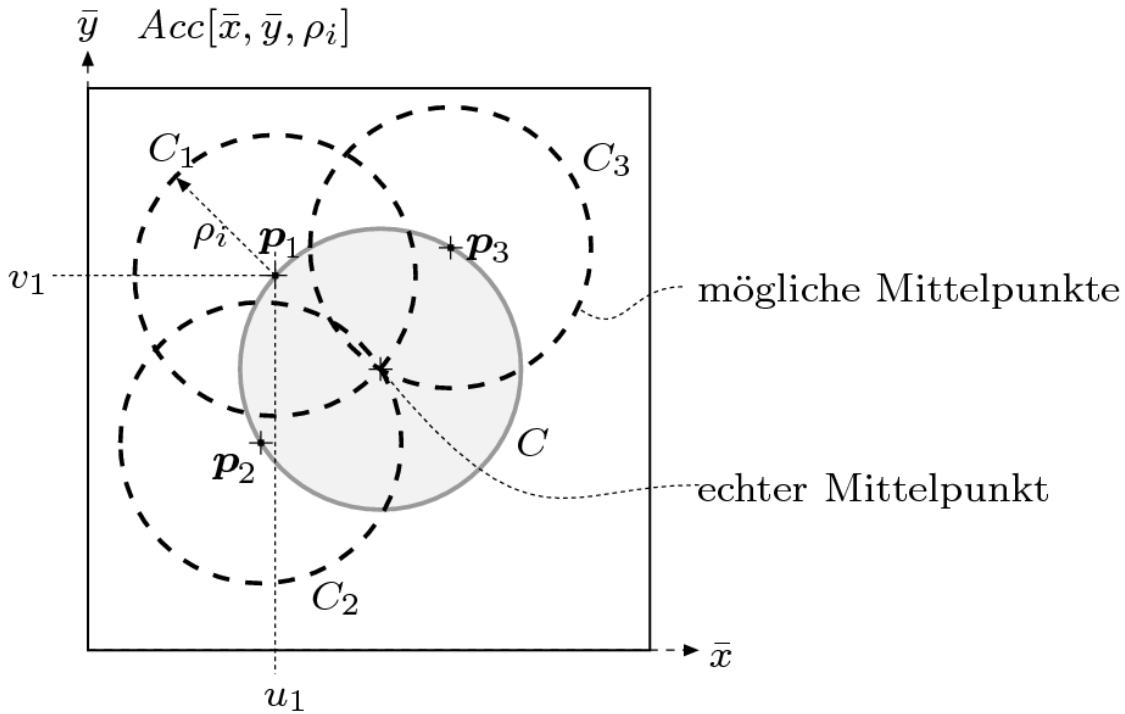
”Brute-Force”-Ansatz

Gesucht wird wieder die Menge aller Kreise, die durch einen gegebenen Bildpunkt $\mathbf{p}(u, v)$ gehen. Leider produziert diese Menge keine einfach zu berechnenden Kurven im Parameterraum.

”Brute-Force”-Ansatz: Teste für jede Zelle im Akkumulator-Array (d.h. jeden Parametersatz), ob er die Kreisgleichung erfüllt:

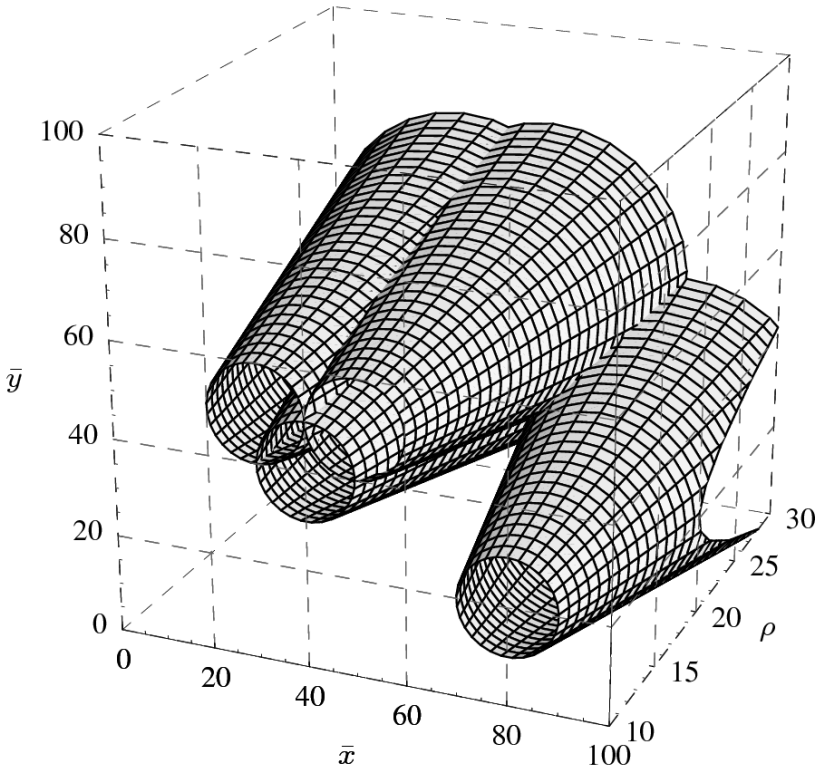
```
1: HoughCircles(I)
2:     Set up a three-dimensional array Acc[x̄, ȳ, ρ] and initialize to 0
3:     for all image coordinates (u, v) do
4:         if I(u, v) is an edge point then
5:             for all (x̄_i, ȳ_i, ρ_i) in the accumulator space do
6:                 if (u − x̄_i)^2 + (v − ȳ_i)^2 = ρ_i^2 then
7:                     Increment Acc[x̄_i, ȳ_i, ρ_i]
8:     MaxCircles ← FindMaxCircles(Acc)   ▷ a list of tuples (x̄_j, ȳ_j, ρ_j)
9:     return MaxCircles.
```

Einfache Kurven für fixen Radius ρ_1



Für einen fixen Radius ρ_1 liegen alle Mittelpunkte von Kreisen durch einen Punkt $\mathbf{p}(u, v)$ ebenfalls auf einem Kreis mit Radius ρ_1 und Mittelpunkt $\mathbf{p}(u, v)$.

Hough-Transformation für Kreise

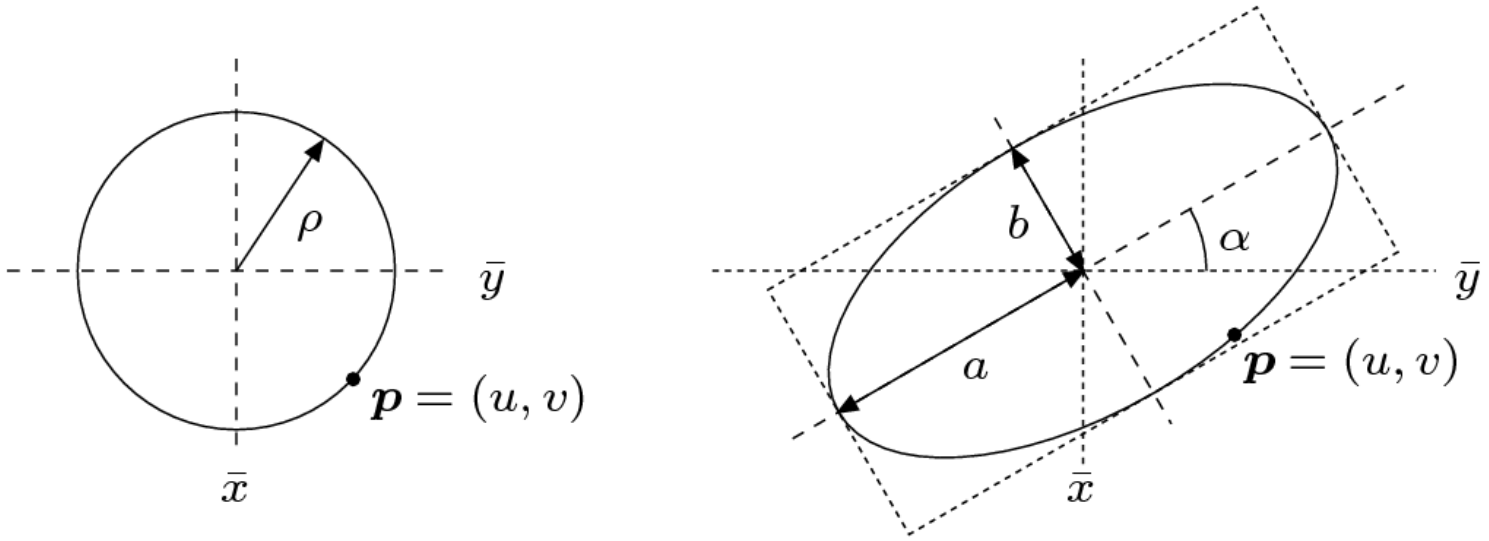


3D-Parameterraum:
$\bar{x}, \bar{y} = 0 \ldots 100$
$\rho = 10 \ldots 30$

Bildpunkte $\boldsymbol{p}_k$:
$\boldsymbol{p}_1 = (30, 50)$
$\boldsymbol{p}_2 = (50, 50)$
$\boldsymbol{p}_3 = (40, 40)$
$\boldsymbol{p}_4 = (80, 20)$

Für jeden neuen Punkt $\mathbf{p}(u, v)$ muß also nicht der gesamte Parameterraum durchsucht und getestet werden, sondern jeweils ein Kreis mit Mittelpunkt $\mathbf{p}(u, v)$ und Radius ρ in jeder Radius-Ebene ρ hochgezählt werden.

Hough-Transformation für Ellipsen



Ellipsen hängen von 5 Parametern ab: x- und y-Position $(\bar{x}, \bar{y})$ des Mittelpunkts und zwei Durchmesser a, b und Orientierung α, d.h. wir benötigen einen 5-dimensionalen Parameterraum. Bei 128 Auflösungsschritten ergibt das 2^{35} Akkumulatorzellen, also bei 4-Byte-Integerzellen 128 GB.

$\Rightarrow$ nicht praktikabel! ($\Rightarrow$ *verallgemeinerte Hough-Transformation*).