03-05-H
-709.53

Echtzeitbildverarbeitung (7)

Prof. Dr. Udo Frese

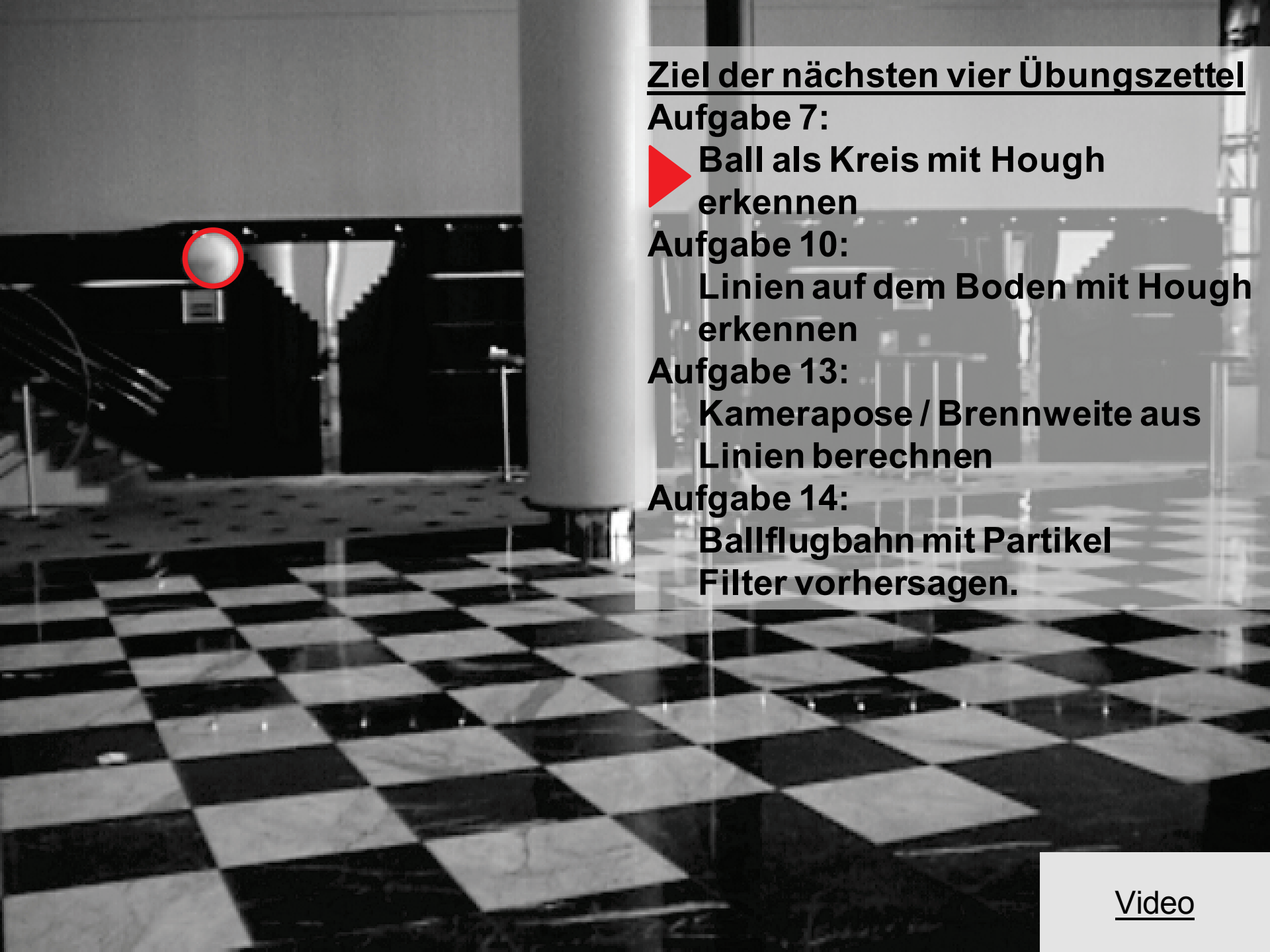
Hough Transformation für Kreise

Zusammenfassung

- **Faltungsoperationen**
  - Lineare, translationsinvariante Abbildung
  - Ergebnispixel ist gewichtete Summe der Pixel in der Umgebung des Eingangspixels
  - Bilder glätten, Kontrast vergrößern, Kanten detektieren
- **Kantendetektion mit dem Sobel Filter (SobelX, SobelY)**
  - SobelX und SobelY geben als Ergebnis einen Vektor
  - Betrag: Kantenstärke
  - Richtung: Richtung der Kante (senkrecht zum Helleren)

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1



Ziel der nächsten vier Übungszettel

Aufgabe 7:

- Ball als Kreis mit Hough erkennen

Aufgabe 10:

- Linien auf dem Boden mit Hough erkennen

Aufgabe 13:

- Kamerapose / Brennweite aus Linien berechnen

Aufgabe 14:

- Ballflugbahn mit Partikel Filter vorhersagen.

Video

- **<u>Frage an das Auditorium</u>: Was ist hier leichter / schwerer?**


Frage an das Auditorium: Was ist hier leichter / schwerer?
- Ball hat teilweise schwachen Kontrast.
- Ball ist weit entfernt.
+ Starke Kanten statt schwacher Linien.

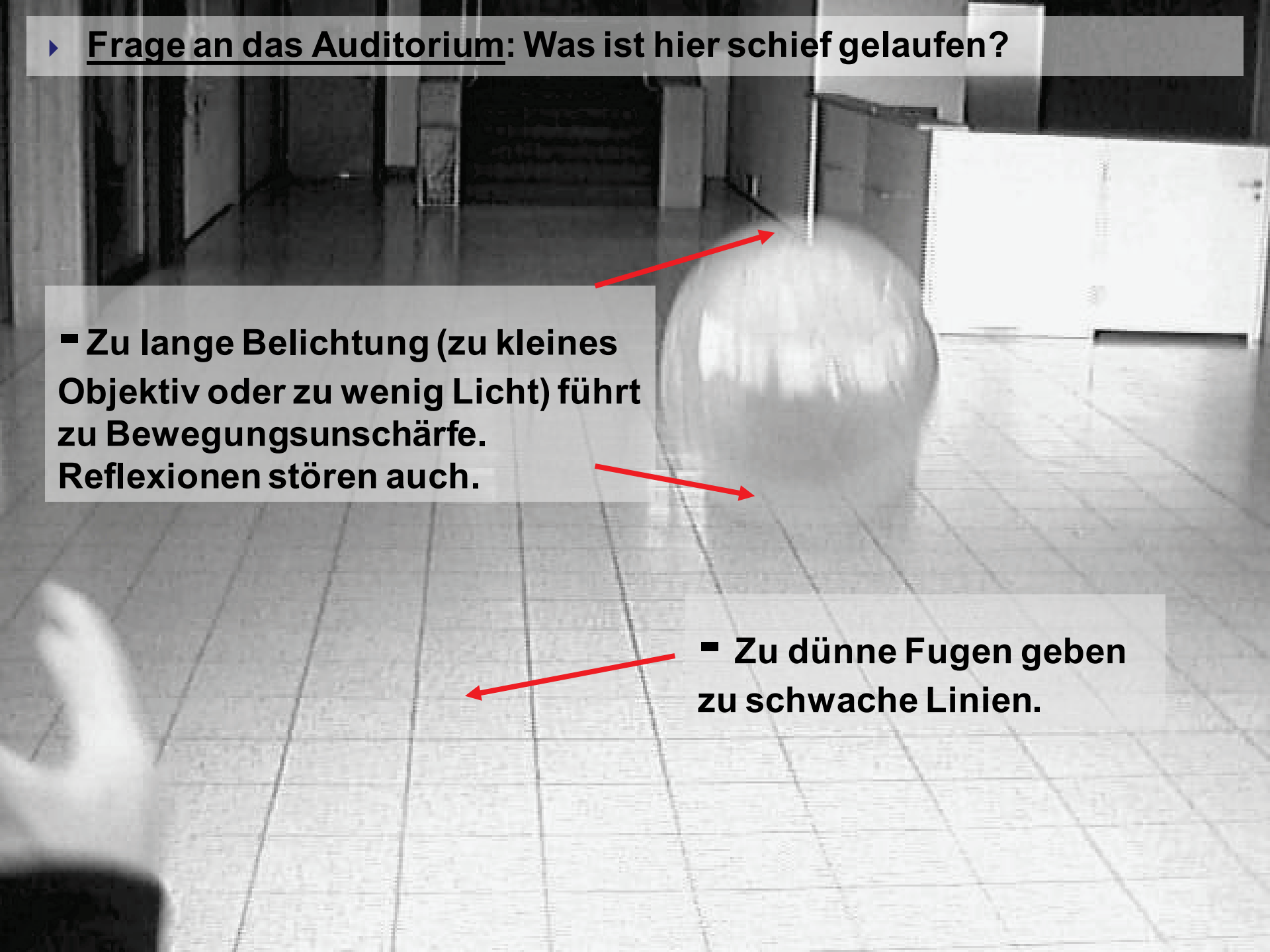
- **Frage an das Auditorium: Was ist hier leichter / schwerer?**

- Zu dünne Fugen geben zu schwache Linien.

+ Ball hat sehr starken Kontrast.

+ Ball ist anfangs nah.

- **Frage an das Auditorium: Was ist hier schief gelaufen?**


Frage an das Auditorium: Was ist hier schief gelaufen?
- Zu lange Belichtung (zu kleines Objektiv oder zu wenig Licht) führt zu Bewegungsunschärfe. Reflexionen stören auch.
- Zu dünne Fugen geben zu schwache Linien.

Hough Transformation für Kreise

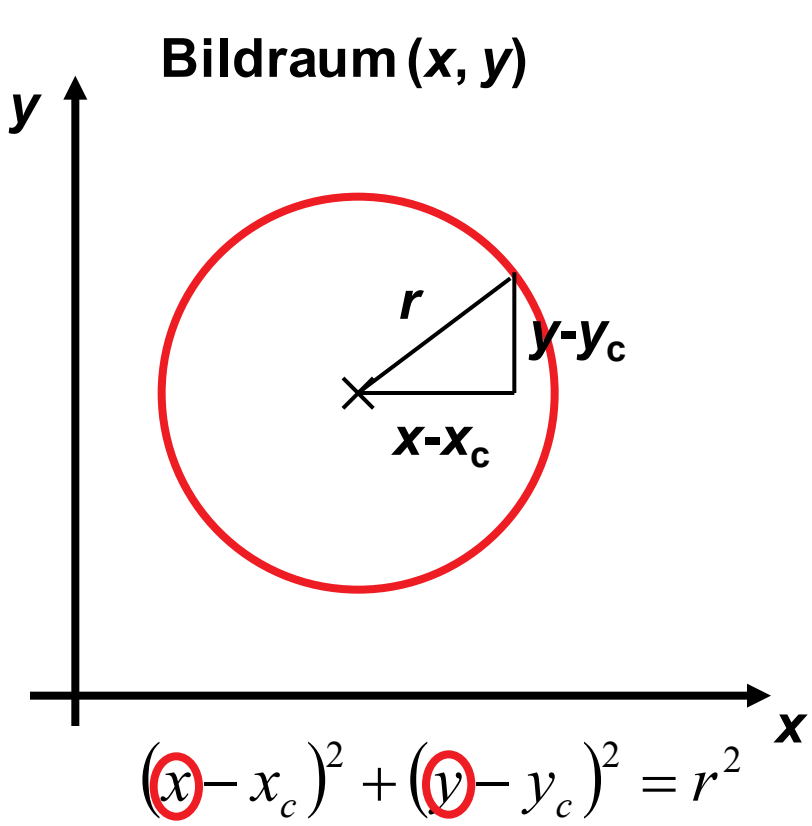
Hough Transformation für Kreise

Grundidee

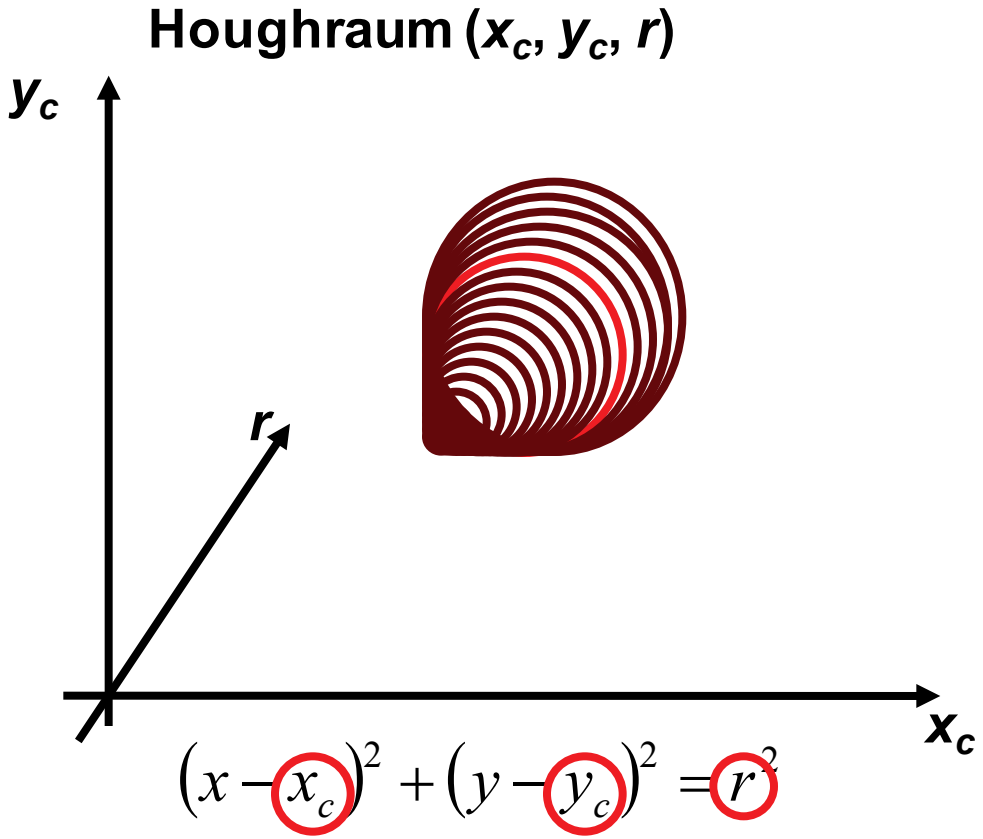
- **Hough Algorithmus sucht Kurven**
  - in bekannter parametrisierter Form
  - mit unbekannten Parametern
  - für Kreise: Mittelpunkt und Radius
- **akkumuliert Evidenz im Parameterraum (Houghraum)**
- ***jeder Kantenpixel ist Evidenz für alle Kurven auf denen er liegt***
  - Houghakkumulator: ein Eintrag pro Parameterkombination
  - Pixel binär (ja/nein) als Kantenpixel klassifizieren (Schwellwert auf Sobellänge)
  - für jeden Kantenpixel alle Akkumulatoreinträge (um 1) erhöhen, deren Kurve durch diesen Pixel geht.
- **Einträge wirklich vorhandener Kurven akkumulieren einem hohen Wert, weil sie von vielen Kantenpixeln erhöht werden**

Hough Transformation für Kreise

Parametrisierung eines Kreises in Mittelpunktsform



$$(x - x_c)^2 + (y - y_c)^2 = r^2$$



$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

Hough Transformation für Kreise

Parametrisierung eines Kreises in Mittelpunktsform

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

- **Struktur des Houghraums (x_c, y_c, r)**
  - `houghImg (xC, yC, r)`
  - Mittelpunkt (x_c, y_c) und Radius
  - (x_c, y_c) haben selbe Dimensionen und Struktur wie das Bild.
  - oder etwas größer für Mittelpunkte außerhalb des Bildes.
  - Intervall zulässiger Radien $r \in [r_{min}, r_{max}]$
- **Akkumulation**
  - laufe durch (x_c, y_c)
  - rechne r als aus
  - erhöhe `houghImg` (x_c, y_c, r) um 1

$$r = \sqrt{(x - x_c)^2 + (y - y_c)^2}$$

- **Problem: 3D Houghraum kostet enormen Speicher und Rechenzeit**

Hough Transformation für Kreise

Hough Transformation für Kreise (Rohfassung)

- **extrem langsam**

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

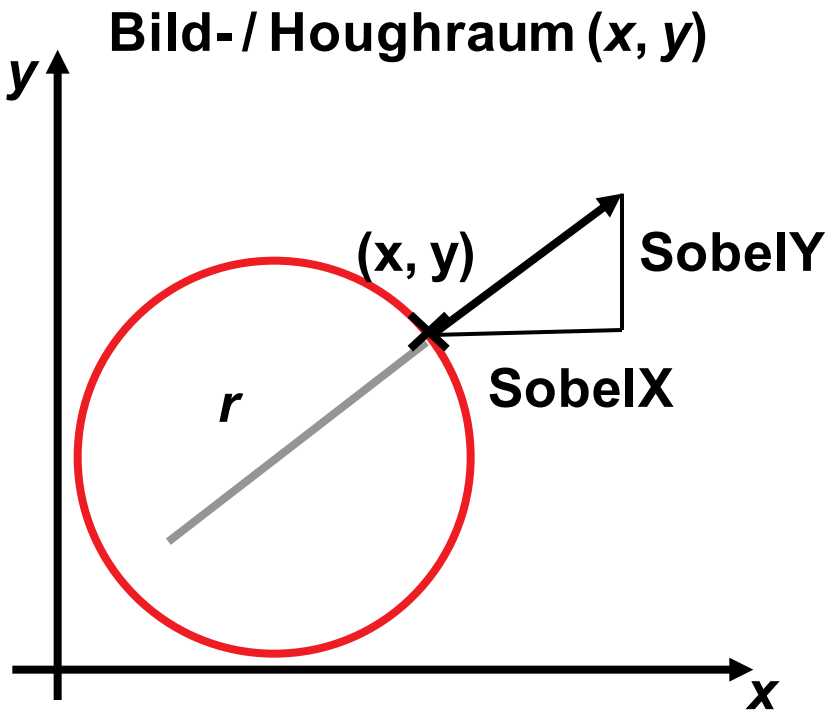
```
addPointToCHA (houghImg, x, y, sobelX, sobelY) {
   for (dy=-RMAX; dy<=RMAX; dy++) for (dx=-RMAX; dx<=RMAX; dx++) {
      r = sqrt(dx*dx + dy*dy);
      houghImg (x+dX, y+dY, r) += 1;
   }
}

circleHough (sobelImg, houghImg, sobelThreshold) {
   houghImg.fill (0)
   for (y=0;y<sobelImg.height;y++)
      for (x=0;x<sobelImg.width;x++) {
         sobelLen = sqrt(sobelImg (x, y).sobelX²+sobelImg (x, y).sobelY²);
         if (sobelLen>sobelThreshold)
            addPointToCHA (houghImg, x, y, sobelX, sobelY);
      }
}
```

Hough Transformation für Kreise

Ausnutzen der Sobelrichtung

- bei idealem Kreis: Kante tangential
- ⇒ Sobel Vektor radial.
- ⇒ (sobelX, sobelY) zeigt zum Mittelpunkt oder von ihm weg.

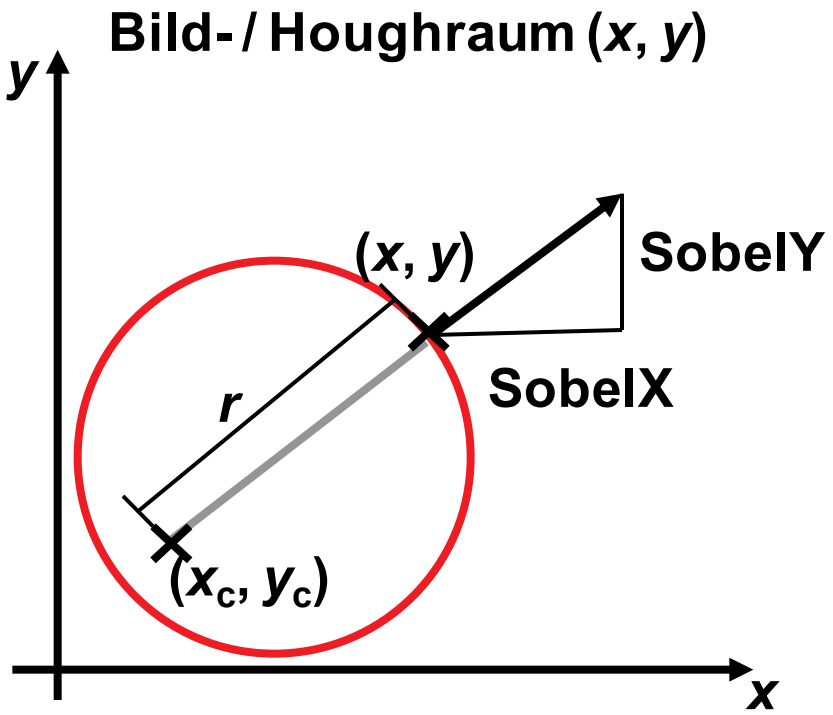


$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

Hough Transformation für Kreise

Ausnutzen der Sobelrichtung

- **effiziente Akkumulation**
  - entlang dem Sobelvektors r Pixel laufen
    $r \in [r_{min} \ldots r_{max}]$
  - $x_c = x + r \cdot \text{sobelX/sobelLen}$
    $y_c = y + r \cdot \text{sobelY/sobelLen}$
  - `houghImg` (x_c, y_c, r) erhöhen
  - dasselbe entgegen dem Sobelvektor
    $x_c = x - r \cdot \text{sobelX/sobelLen}$
    $y_c = y - r \cdot \text{sobelY/sobelLen}$
- **nur $2(r_{max}-r_{min}+1)$ statt $4r_{max}^2$ Einträge**



$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

Hough Transformation für Kreise

Hough Transformation Kreise (1. optimierte Fassung)

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

```
addPointToCHA (houghImg, x, y, sobelX, sobelY) {
   sobelLen = sqrt(sobelX²+sobelY²);
   for (r=RMIN; r<=RMAX; r++) {
      // Entlang Normalenvektor
      xc = x + r*((double) sobelX)/sobelLen;
      yc = y + r*((double) sobelY)/sobelLen;
      houghImg (xc, yc, r) += 1;
      // Entgegen Normalenvektor
      xc = x - r*((double) sobelX)/sobelLen;
      yc = y - r*((double) sobelY)/sobelLen;
      houghImg (xc, yc, r) += 1;
   }
}
```

Randüber-schreitung (später lösen)

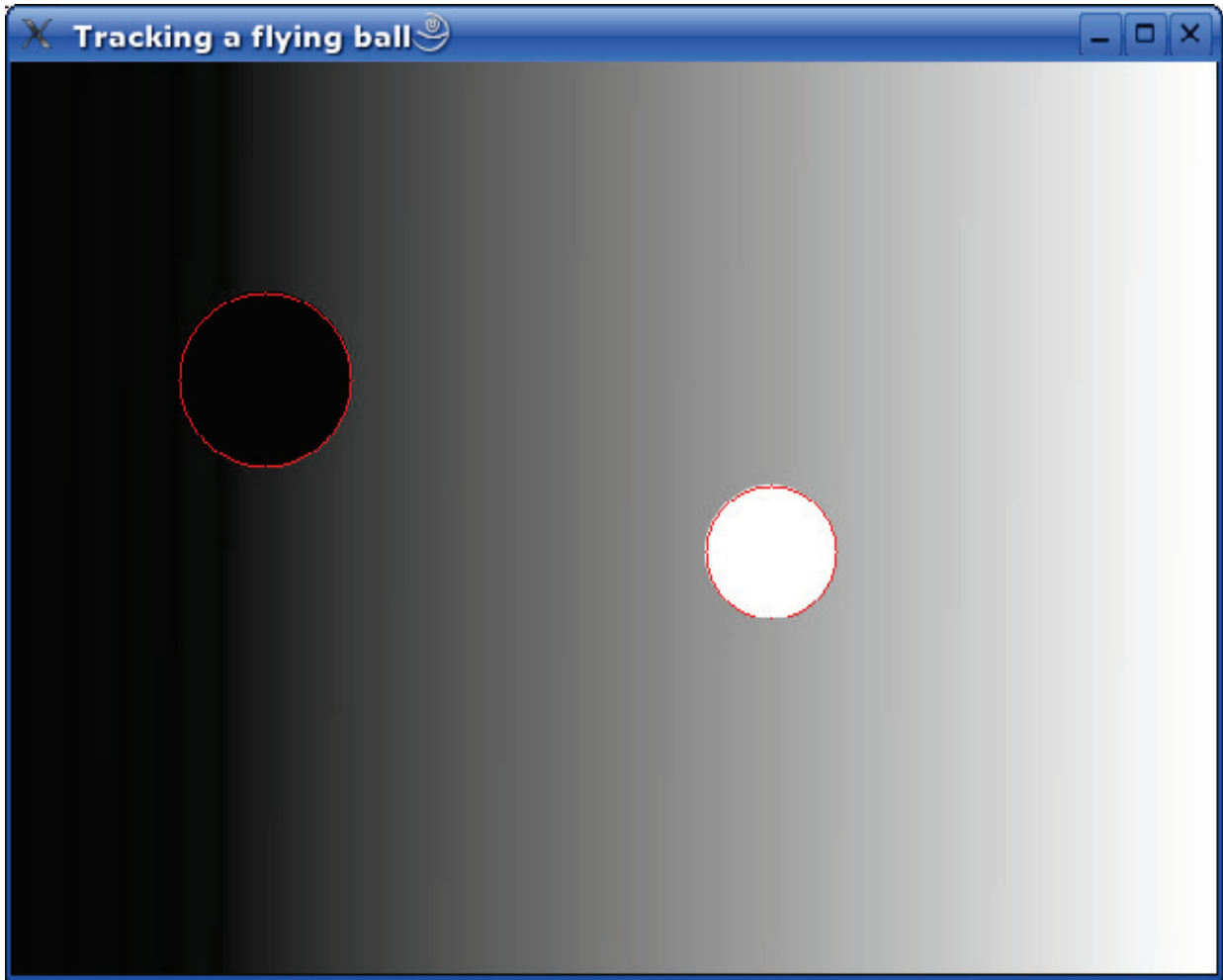
Hough Transformation für Kreise

Houghraum auf 2D reduzieren

- **Problem: 3D Houghraum benötigt viel Speicherplatz und Rechenzeit**
- **Lösung: in Bildebene (x_c, y_c) projizieren**
  - ⇒ `houghImg(xc, yc)` statt `houghImg(xc,yc,r)`
  - Vorauswahl der Kreismittelpunkte
  - ⇒Kreise mit demselben Mittelpunkt überlagern sich
- **wesentlich effizienter**
- **später auf den maximalen (x_c, y_c) Punkten Suche nach maximalem *r*.**
- **Problem: mehr Störungen im Houghraum, weil auf jeden richtigen Radius r_{max}-r_{min} falsche kommen.**

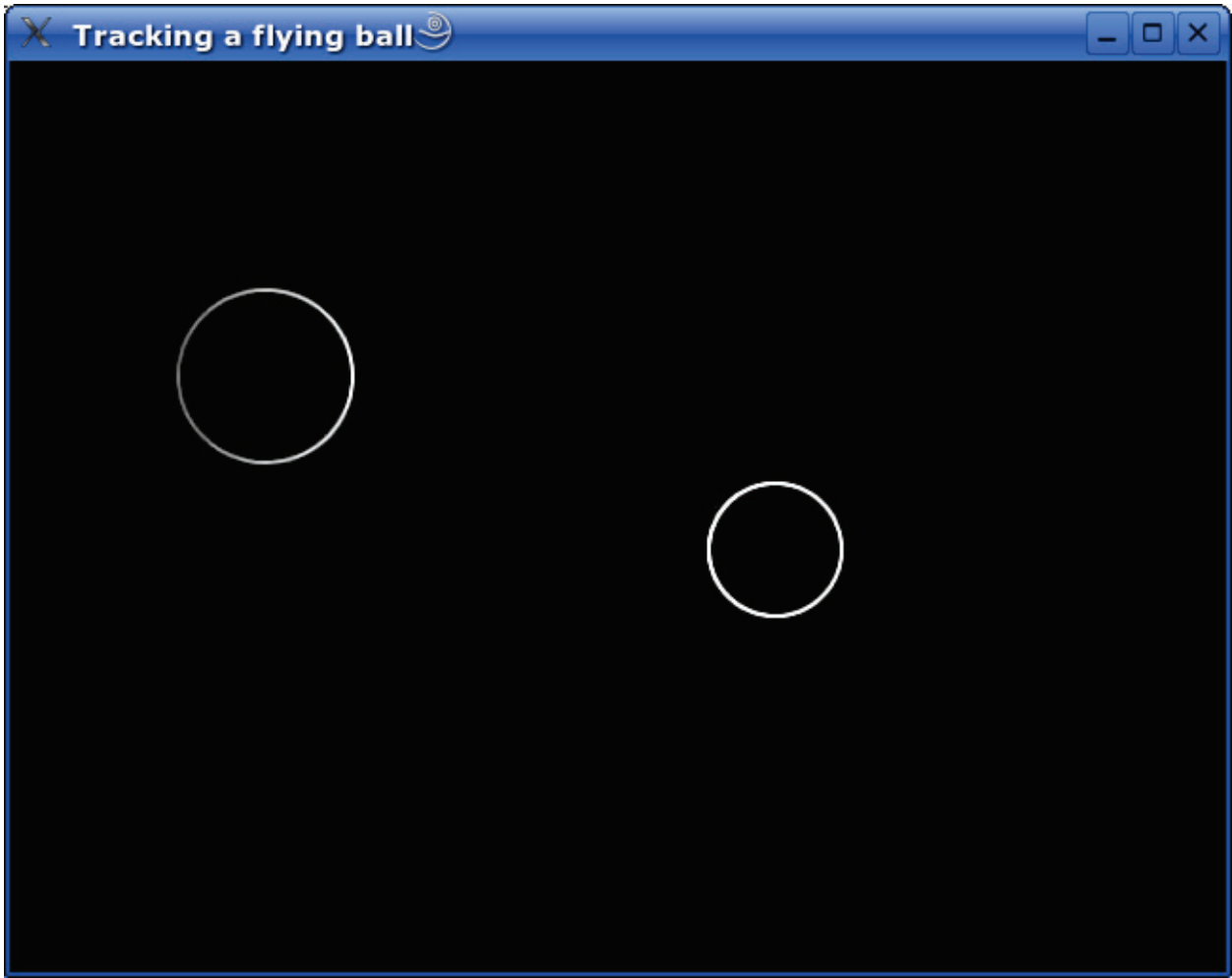
Hough Transformation für Kreise

Testbild: erkannte Kreise



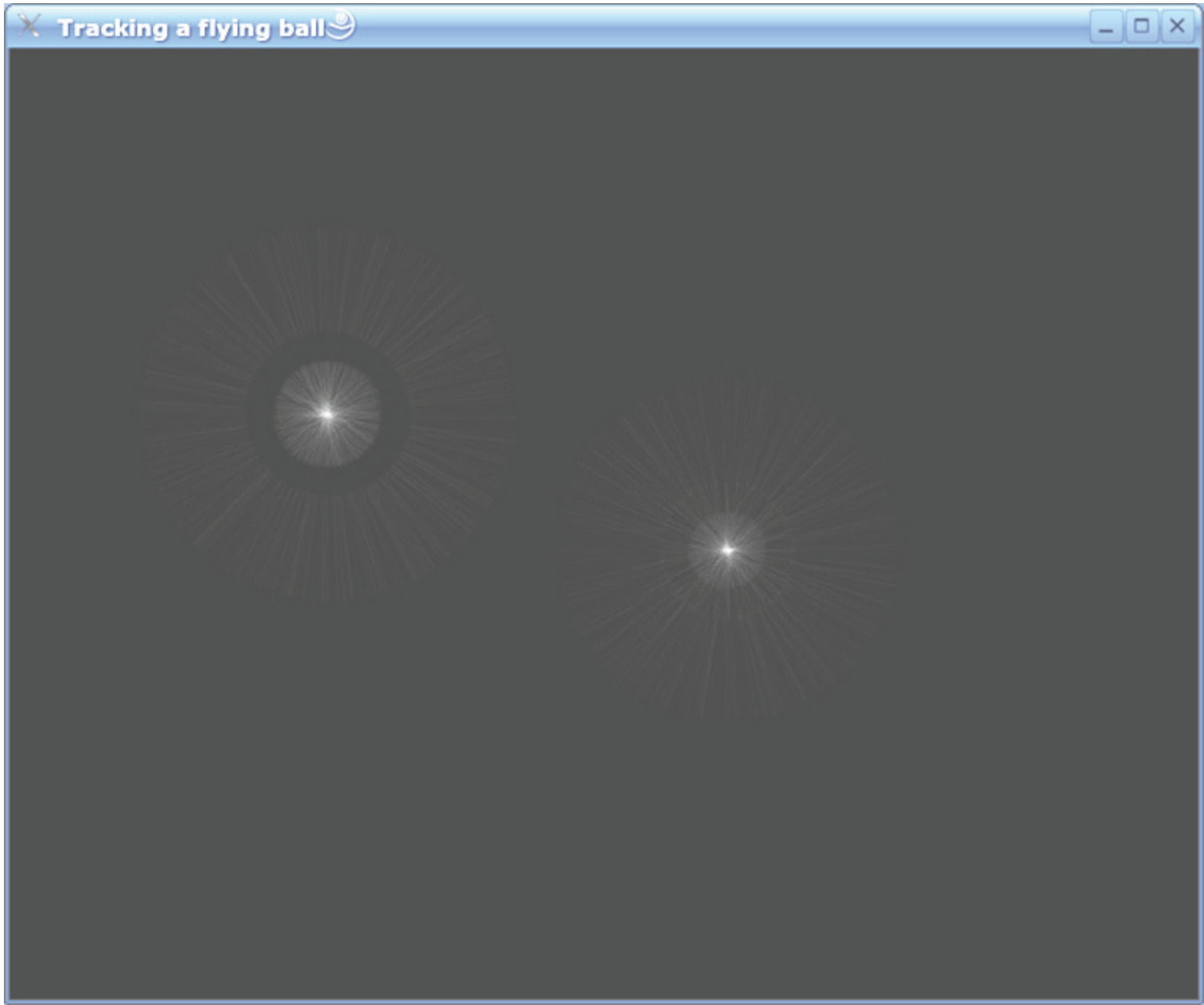
Hough Transformation für Kreise

Testbild: Sobellänge



Testbild: Houghraum (aufgehellt)

Frage an das Auditorium: Könnt Ihr das Bild erklären?



Hough Transformation für Kreise

Effizienz durch Tabellen (LUT)

- **Problem:**
  Aufwändige Rechnungen und langsame `double`/`int` Konversion.

```
addPointToCHA (houghImg, x, y, sobelX, sobelY) {
   sobelLen = sqrt(sobelX²+sobelY²);
   for (r=RMIN; r<=RMAX; r++) {
      // Entlang Normalenvektor
      xc = x + r*((double) sobelX)/sobelLen;
      yc = y + r*((double) sobelY)/sobelLen;
      houghImg (xc, yc) += 1;
      // Entgegen Normalenvektor
      xc = x - r*((double) sobelX)/sobelLen;
      yc = y - r*((double) sobelY)/sobelLen;
      houghImg (xc, yc) += 1;
   }
}
```

Effiziente Hough-Transformation

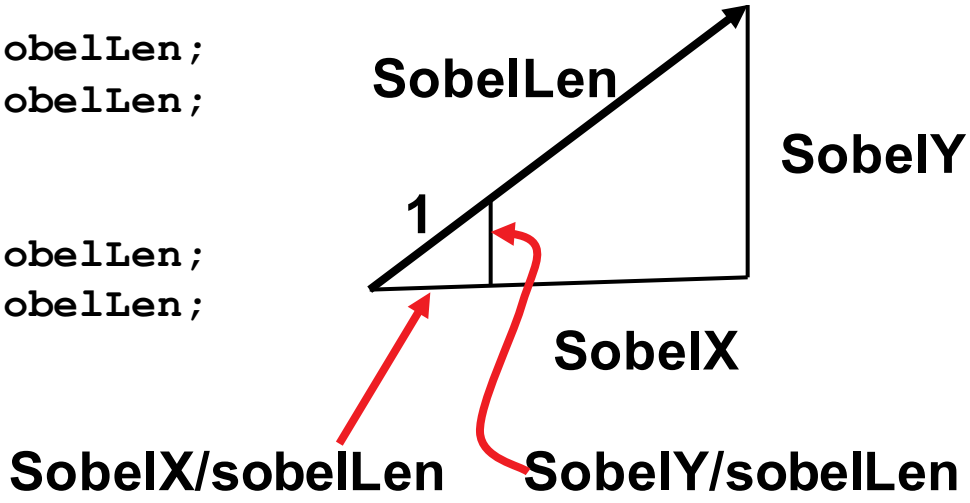
Effizienz durch Tabellen (LUT)

- **für: komplizierte Rechnungen mit wenigen Variablen**
- **Beispiel: Farbklassifikation aus R, G, B**
- **Idee: Tabelle mit Ergebnis für jede Kombination der Variablen**
- **mehrere Ergebnisse zur selben Variablenkombination möglich**
- **gut, wenn Variablen schon diskret / diskretisiert sind**
- **Zusatzrechnungen in Tabelle integrieren**
  - Diskretisierung
  - Normalisierung
  - Beschränkung des Ergebnisses
  - t.w. Adressberechnungen für z.B. Bildzugriff

Hough Transformation für Kreise

- **Frage an das Auditorium: Wo kann man LUT einsetzen?**

```
addPointToCHA (houghImg, x, y, sobelX, sobelY) {
   sobelLen = sqrt(sobelX²+sobelY²);
   for (r=RMIN; r<=RMAX; r++) {
      // Entlang Normalenvektor
      xc = x + r*((double) sobelX)/sobelLen;
      yc = y + r*((double) sobelY)/sobelLen;
      houghImg (xc, yc) += 1;
      // Entgegen Normalenvektor
      xc = x - r*((double) sobelX)/sobelLen;
      yc = y - r*((double) sobelY)/sobelLen;
      houghImg (xc, yc) += 1;
   }
}
```



Hough Transformation für Kreise

LUT 1: (sobelX, sobelY)→ (sobelLen, sobelAngle)

```
addPointToCHA (houghImg, x, y, sobelX, sobelY) {
   sobelLen = sqrt(sobelX²+sobelY²);
   for (r=RMIN; r<=RMAX; r++) {
      // Entlang Normalenvektor
      xc = x + r*((double) sobelX)/sobelLen;
      yc = y + r*((double) sobelY)/sobelLen;
      houghImg (xc, yc) += 1;
      // Entgegen Normalenvektor
      xc = x - r*((double) sobelX)/sobelLen;
      yc = y - r*((double) sobelY)/sobelLen;
      houghImg (xc, yc) += 1;
   }
}
```

LUT 2: (sobelAngle, r) → (relative Adresse)

((double) sobelX)/sobelLen == cos(sobelAngle)

((double) sobelY)/sobelLen == sin(sobelAngle)

houghRef = &houghImg (x,y), houghRef[(relative Adresse)]

Hough Transformation für Kreise

Houghtransformation für Kreise (2. optimierte Fassung)

```
addPointToCHA (houghImg, x, y, sobelX, sobelY) {
   sobelAngle = lut1(sobelX, sobelY).angle;
   houghRef = &houghImg(x, y)
   for (r=RMIN; r<=RMAX; r++) {
      relAddr = lut2(sobelAngle, r);
      // Entlang Normalenvektor
      houghRef[ relAddr] += 1;
      // Entgegen Normalenvektor
      houghRef[-relAddr] += 1;
   }
}
```

Hough Transformation für Kreise

Houghtransformation für Kreise (2. optimierte Fassung)

- 2 Probleme zu lösen

```
addPointToCHA (houghImg, x, y, sobelX, sobelY) {
    sobelAngle = lut1(sobelX, sobelY).angle;
    houghRef = &houghImg(x, y)
    for (r=RMIN; r<=RMAX; r++) {
        relAddr = lut2[sobelAngle, r];
        // Entlang Normalenvektor
        houghRef[ relAddr] += 1;
        // Entgegen Normalenvektor
        houghRef[-relAddr] += 1;
    }
}
```

Ganze Zahl für Tabellenzugriff nötig

Randüber-schreitung

Hough Transformation für Kreise

Festkommaarithmetik

- reelle Zahlen als ganzen Zähler mit festem Nenner darstellen
- oft 2^i als Nenner
- Beispiel: 0.5 $\rightarrow$ 128/256, nur 128 gespeichert
- vermeidet `float` oder `double` (nicht primärer Vorteil).
- vermeidet Konversionen `int-float/double`.
- ermöglicht direkten Tabellenzugriff.
- hier: `sobelAngle` auf [0.. π) auf [0..256) skalieren

Hough Transformation für Kreise

- **Frage an das Auditorium: Wie würde untenstehender Code in Festkomma-Arithmetik aussehen (zur Basis $1/256=1/2^8$)?**

```
double scalar (double x0, double y0, double x1, double y1)
{
   return x0*x1+y0*y1;
}
```

Hough Transformation für Kreise

- **Frage an das Auditorium: Wie würde untenstehender Code in Festkomma-Arithmetik aussehen (zur Basis $1/256=1/2^8$)?**

```
double scalar (double x0, double y0, double x1, double y1)
{
   return x0*x1+y0*y1;
}

int scalar (int x0, int y0, int x1, int y1)
{
   return (x0*x1+y0*y1)>>8;
}
```

Hough Transformation für Kreise

Implementierung der LUTs

- **LUT 1: `(sobelX, sobelY)` → `(sobelLen, sobelAngle)`**
  - Tabelle (vector) von Objekten, nicht mehrere Tabellen
  - `sobelLen = sqrt(sobelX`^{2}`+sobelY`^{2}`)`
  - `sobelAngle = atan2 (sobelY, sobelX)`
  - Auf [0.. π) bzgl. der Periodizität normalisieren
  - Auf [0.. π) auf [0..256) skalieren (Festkomma), so dass halbe Drehung 256 ist.
  - [0..256) heisst 0 inclusive, 256 exclusive
  - Wertebereich für `sobelX`, `sobelY`: [-1020..1020]
  - Eine Zeile: 2^{11}, dadurch Multiplikation `<<11`, Offset `1024`
  - **Achtung**: Klammern bei <<
  - Zugriff: `sobelTab[((sobelY+1024)<<11) + (sobelX+1024)]`
  - `+1024` herausziehen mit `sobelCenterTab[(sobelY<<11)+sobelX]`

Hough Transformation für Kreise

Implementierung der LUTs

- **LUT 2: `(sobelAngle, r)` → `(relative Adresse)`**
  - `dX = r*cos(sobelAngle)`
  - `dY = r*sin(sobelAngle)`
  - `sobelAngle` auf [0.. π) normalisiert und auf [0..255] skaliert (Festkomma).
  - `relative Adresse = dX+widthStep*dY`
  - Achtung: `houghImg->widthStep` in bytes, pointer und `widthStep` in unsigned short.
    ⇒ `widthStep=houghImg->widthStep/sizeof(unsigned short)`.

Hough Transformation für Kreise

Houghtransformation für Kreise (2. optimierte Fassung)

- 1 Problem zu lösen

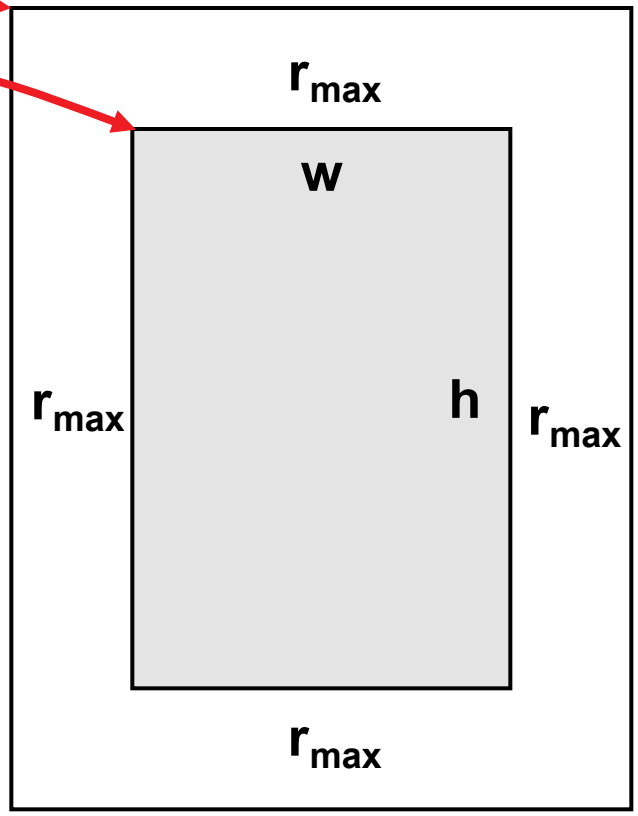
```
addPointToCHA (houghImg, x, y, sobelX, sobelY) {
    sobelAngle = lut1(sobelX, sobelY).angle;
    houghRef = &houghImg(x, y)
    for (r=RMIN; r<=RMAX; r++) {
        relAddr = lut2[sobelAngle, r];
        // Entlang Normalenvektor
        houghRef[ relAddr] += 1;
        // Entgegen Normalenvektor
        houghRef[-relAddr] += 1;
    }
}
```

Randüber-schreitung

Hough Transformation für Kreise

houghImage->imageData

houghOrigin



- Hough-Raum wird um $r_{max} \times r_{max}$ vergrößert um
- Randtests zu vermeiden
- Kreise deren Mittelpunkte ausserhalb des Bildes liegen zu erkennen
- $r_{max}+r_{max}$*`houghWidthStep` addieren
- oder `houghOrigin` nutzen

Hough Transformation für Kreise

Effiziente Implementierung der Hough Transformation für Kreise

- **Laufzeiger statt Koordinaten.**
- **Tabellen**
  - LUT1: Länge und Richtung des Sobelvektors
  - LUT2: für jede Sobel Richtung und jedes r die Adresse des Pixels (x_c, y_c) relativ zu (x, y).
- **Randtest (Clipping)**
  - Houghraum um r_{max} vergrößern als Bild.
  - $\Rightarrow$ kein Randtest

Hough Transformation für Kreise

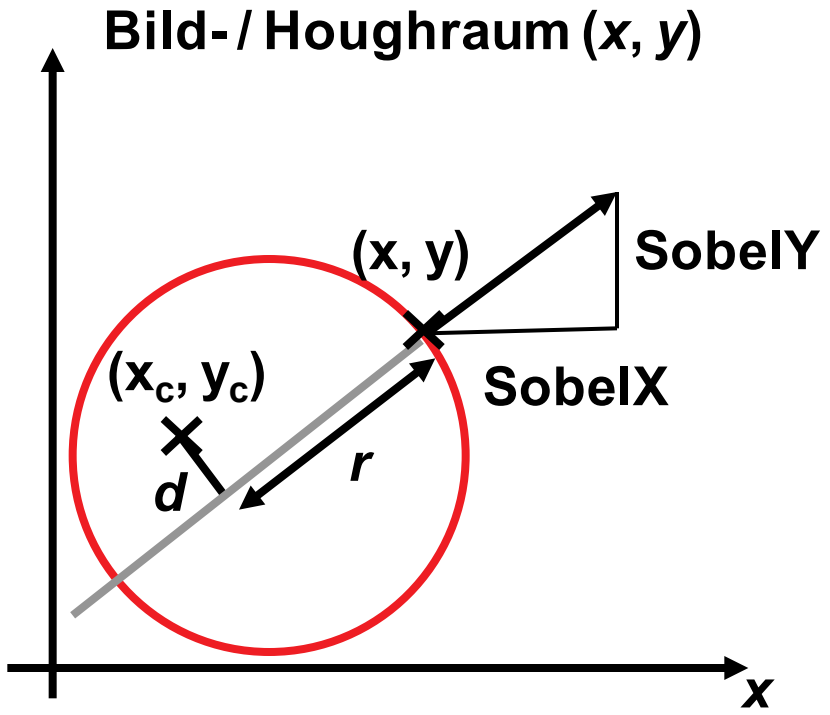
Suchen der Kreise im Houghraum

- **Durchsuche Houghraum**
  - Pixel über Schwellwert
  - und lokales Maximum innerhalb von ($\pm r_{max} \times \pm r_{max}$)

Hough Transformation für Kreise

Suchen der Kreise im Houghraum

- **Suche nach Radius *r***
- **baue 1D Houghraum über *r* auf.**
- **für jedes Maximum (x_c, y_c) durchlaufe ($\pm r_{max} \times \pm r_{max}$) Umgebung**
- **zähle nur die Kantenpixel, die auch im 2D Houghraum in (x_c, y_c) akkumuliert worden sind.**
  - $\Rightarrow$ nur die (x,y) für die d<=1 ist.
  - akkumuliere in hough[r].



$$r = \left|\cos\alpha\left(x - x_c\right) + \sin\alpha\left(y - y_c\right)\right|$$

$$d = \left|-\sin\alpha\left(x - x_c\right) + \cos\alpha\left(y - y_c\right)\right|$$

Hough Transformation für Kreise

Houghtransformation für Kreise (2. optimierte Fassung)

```
houghOnRadiusWithFixedCenter (highestR, sobelImg, xC, yC, sobelThreshold)
{
   for (dy=-RMAX; dy<=RMAX; dy++)
      for (dx=-RMAX; dx<=RMAX; dx++) {
         (sobelX, sobelY) = sobelImg[yC+dY][xC+dX]
         sobelLen = sqrt(sobelX²+sobelY²);
         sobelAngle = arctan2 (sobelY, sobelX);
         r = fabs( cos(sobelAngle)* dx + sin(sobelAngle)*dy);
         d = fabs(-sin(sobelAngle)* dx + cos(sobelAngle)*dy);
         if (d<=1 && sobelLen>sobelThreshold) {
            houghImg[r] += 1;
         }
      }
   highestR = RMIN;
   for (r=RMIN; r<=RMAX; r++)
      if (houghImg[r]>houghImg[highestR]) highestR = r;
}
```

Hough Transformation für Kreise

Houghtransformation für Kreise (2. optimierte Fassung)

```
houghOnRadiusWithFixedCenter (highestR, sobelImg, xC, yC, sobelThreshold)
{
   for (dy=-RMAX; dy<=RMAX; dy++)
      for (dx=-RMAX; dx<=RMAX; dx++) {
         (sobelX, sobelY) = sobelImg[yC+dY][xC+dX]
         sobelLen = sqrt(sobelX²+sobelY²);
         sobelAngle = arctan2 (sobelY, sobelX);
         r = fabs( cos(sobelAngle)*dx + sin(sobelAngle)*dy);
         d = fabs(-sin(sobelAngle)*dx + cos(sobelAngle)*dy);
         if (d<=1 && sobelLen>sobelThreshold) {
            houghImg[r] += 1;
         }
      }
   highestR = RMIN;
   for (r=RMIN; r<=RMAX; r++)
      if (houghImg[r]>houghImg[highestR]) highestR = r;
}
```

LUT1

LUT1
& Festkomma

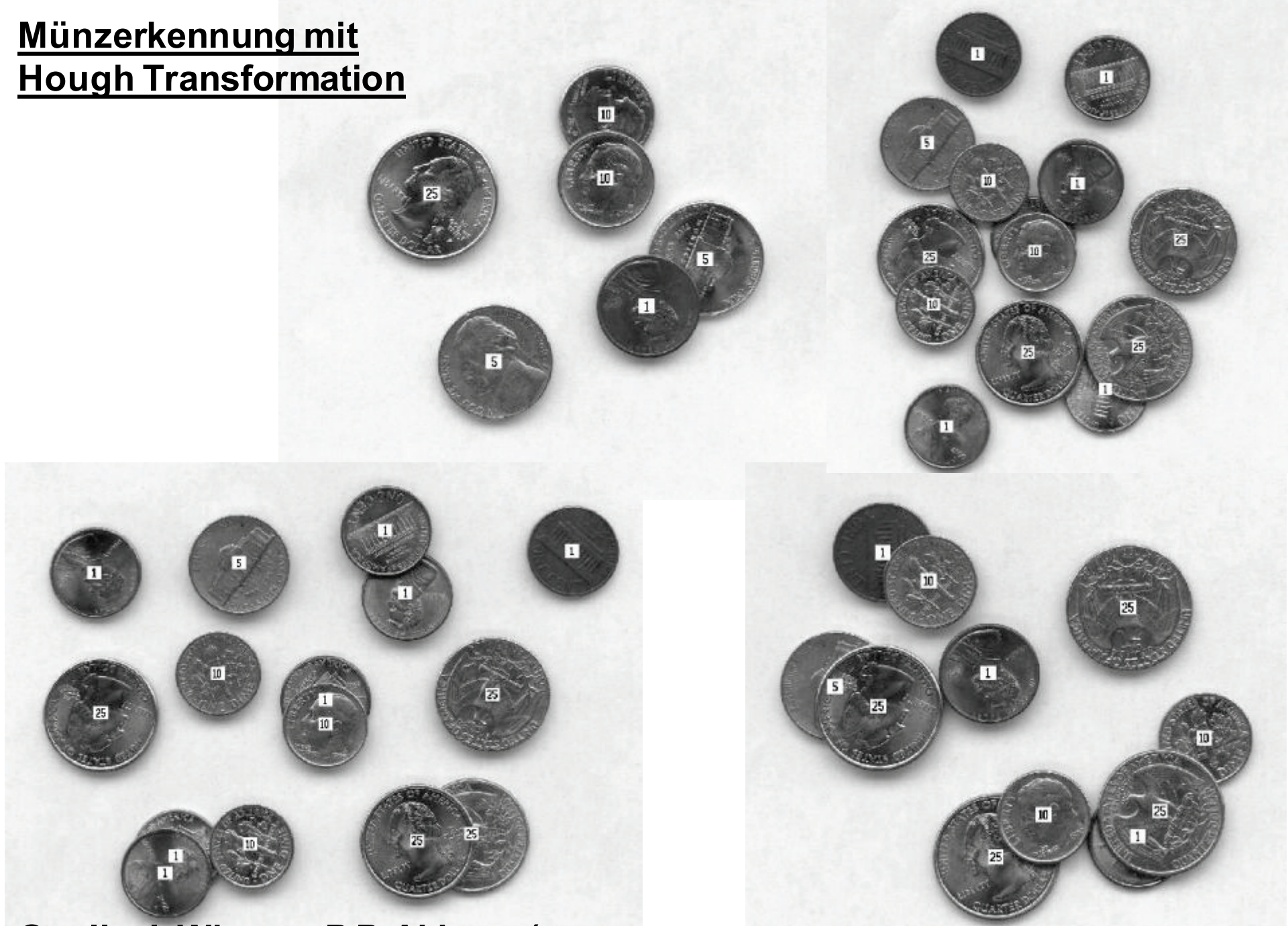
Hough Transformation für Kreise

Houghtransformation für Kreise (2. optimierte Fassung)

```
houghOnRadiusWithFixedCenter (highestR, sobelImg, xC, yC, sobelThreshold)
{
   for (dy=-RMAX; dy<=RMAX; dy++)
      for (dx=-RMAX; dx<=RMAX; dx++) {
         (sobelX, sobelY) = sobelImg[yC+dY][xC+dX]
         sobelLen = sqrt(sobelX²+sobelY²);
         sobelAngle = arctan2 (sobelY, sobelX);
         r = fabs( cos(sobelAngle)*dx + sin(sobelAngle)*dy);
         d = fabs(-sin(sobelAngle)*dx + cos(sobelAngle)*dy);
         if (d<=1 && sobelLen>sobelThreshold) {
            houghImg[r] += 1;
         }
      }
   highestR = RMIN;
   for (r=RMIN; r<=RMAX; r++)
      if (houghImg[r]>houghImg[highestR]) highestR = r;
}
```

Clipping beachten

Münzerkennung mit Hough Transformation



Quelle: I. Wiemer, P.P. Akkam, (www.angelfire.com/vt2/project/)

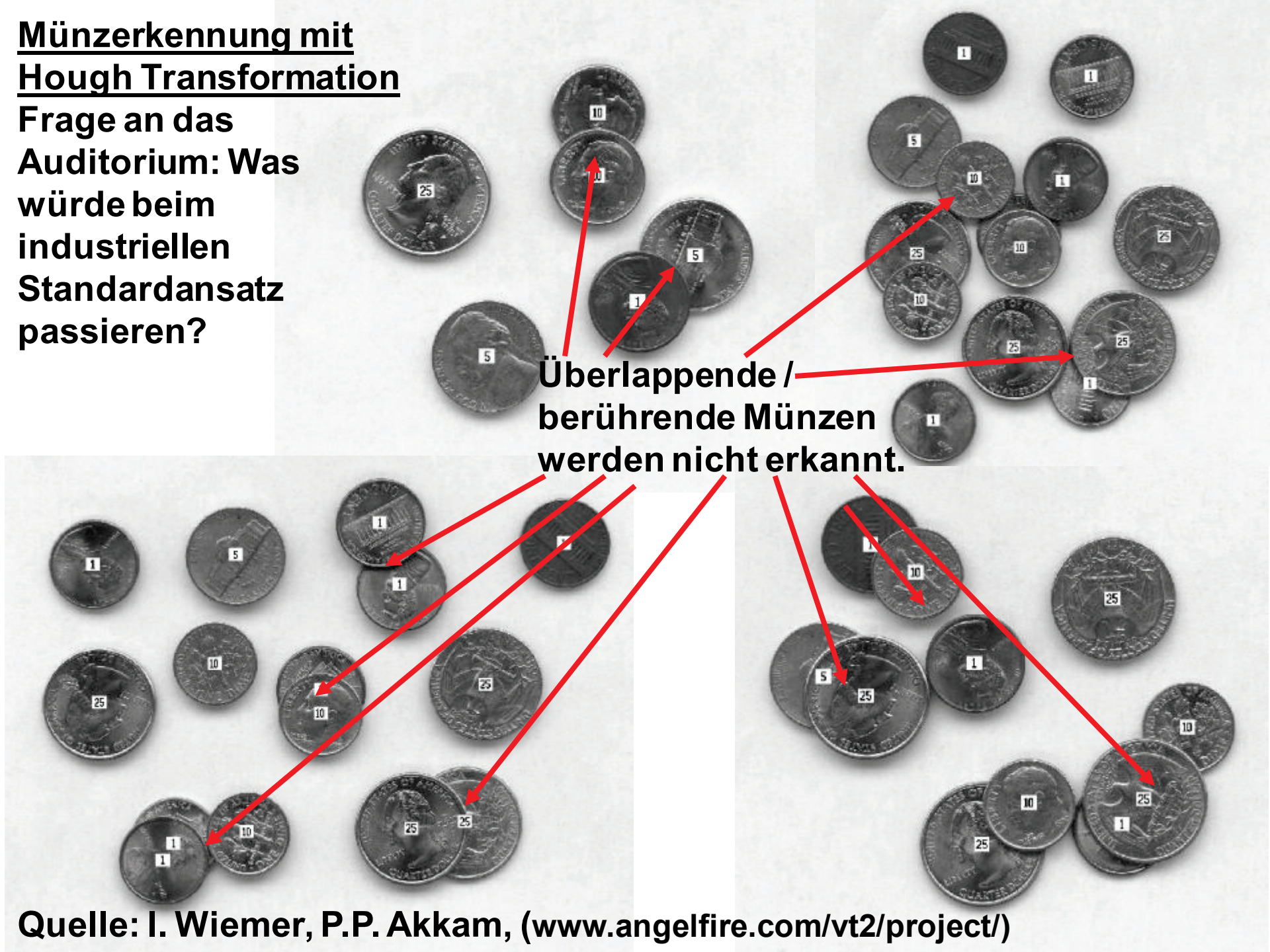
Münzerkennung mit Hough Transformation

Frage an das Auditorium: Was würde beim industriellen Standardansatz passieren?

Quelle: I. Wiemer, P.P. Akkam, (www.angelfire.com/vt2/project/)

Münzerkennung mit Hough Transformation

Frage an das Auditorium: Was würde beim industriellen Standardansatz passieren?



Quelle: I. Wiemer, P.P. Akkam, (www.angelfire.com/vt2/project/)

Zusammenfassung

- **Hough-Transformation**
  - Suche parametrisierte Kurven, z.B. Geraden, Kreise
  - Hough Akkumulator im Parameterraum
  - Für jeden Pixel mit hinreichendem Kontrast erhöhe alle Akkumulatorpixel, deren Kurven diesen Pixel durchlaufen
- **Hough Transformation für Kreise**
  - Houghraum beschreibt Kreise in Mittelpunktsform (x_c, y_c, r)
  - Ausnutzen des Gradienten: (x_c,y_c) = (x, y) +/-r (sobelX, sobelY)/sobelLen
  - Projektion in (x_c, y_c), also weglassen von r weil 3D Houghraum sehr groß.
  - Für jeden (x_c, y_c) Mittelpunkt mit hinreichend großem Hough Wert eine 1D. Hough Transformation bzgl. r durchführen.
- **Derart technisch verwickelte Optimierung sind nur sinnvoll für Teilroutinen die sehr oft ausgeführt werden (z.B. jeden Pixel)**