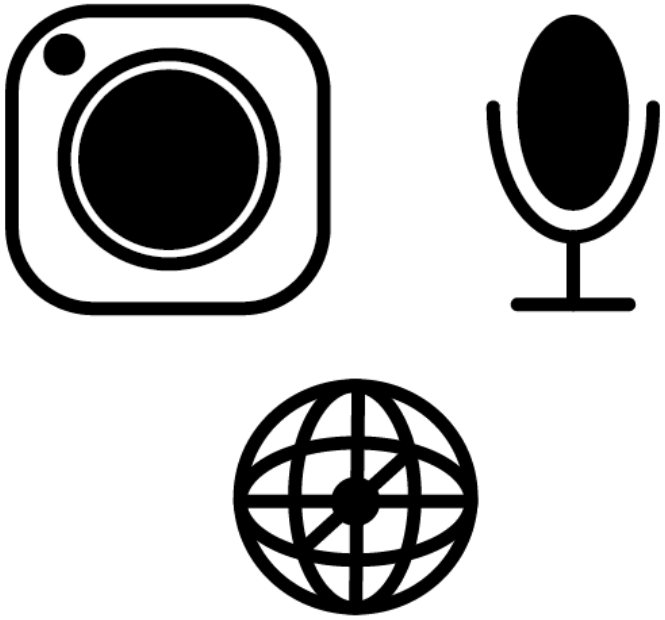


(9.-14.12.24)

Sensordatenverarbeitung

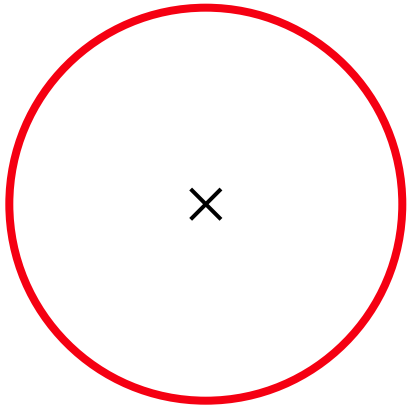
HOUGHTRANSFORMATION (9A)

Zeitplan

Nr.	Thema
1	Einleitung; einführende Beispiele
2	Datenaufnahme; Audio-Datenaufnahme
3	Bild-Datenaufnahme
4	Farbe, Segmentierung, Segmentierungsgetriebene BV
5	Audiosignal, 1D Frequenzraum, Fouriertransformation
6	Koordinatensysteme; Bewegungs-Datenaufnahme
7	2D Frequenzraum, 2D Filter
8	Kanten, SdV-Paradigmen, direkte Bildmerkmale
9	Houghtransformation, Bewegungsmerkmale
10	Audiomerkmale
11	Klassifizierungsalgorithmen
12	Entwicklung und Evaluation sensorbasierter Systeme
13	Bayes-Schätzung & Bayes-Filter
14	Anwendungsbeispiele

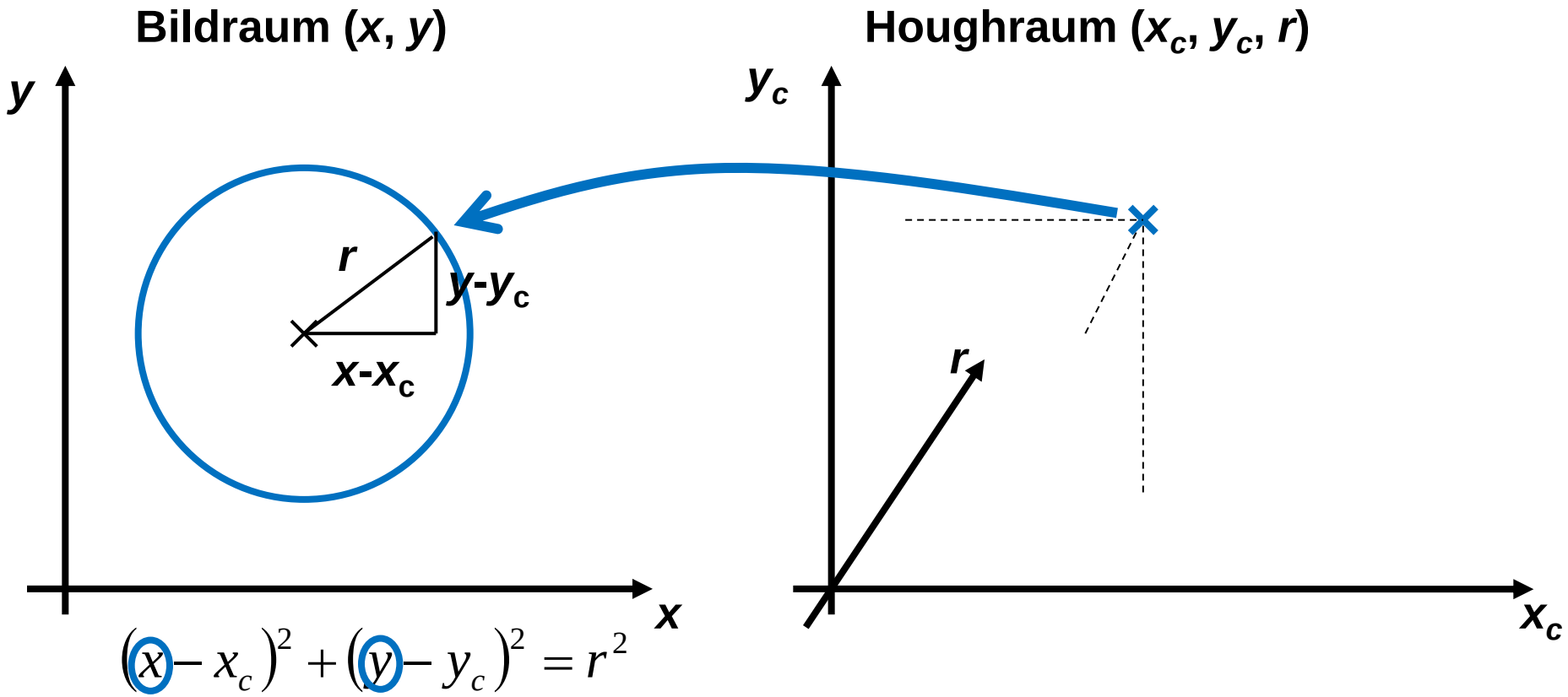
Hough-Transformation für Kreise

Grundidee der Hough-Transform.

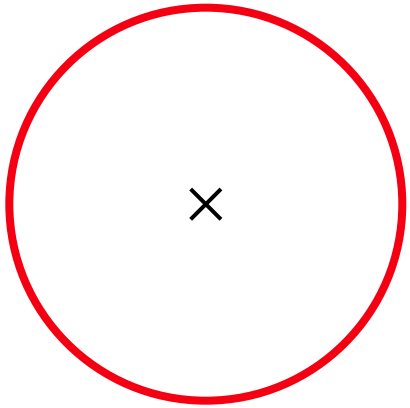


- Hough Algorithmus sucht Kurven
  - in bekannter parametrisierter Form
  - mit unbekannten Parametern
  - für Kreise: Mittelpunkt und Radius
- Eine Form von Merkmalsextraktion
  - Aber: meist zu wenig reichhaltig für ML-Klassifizierung
  - Eher: Man weiß schon was es ist und sucht wo es ist
- Akkumuliert 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 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

Houghraum für Kreise

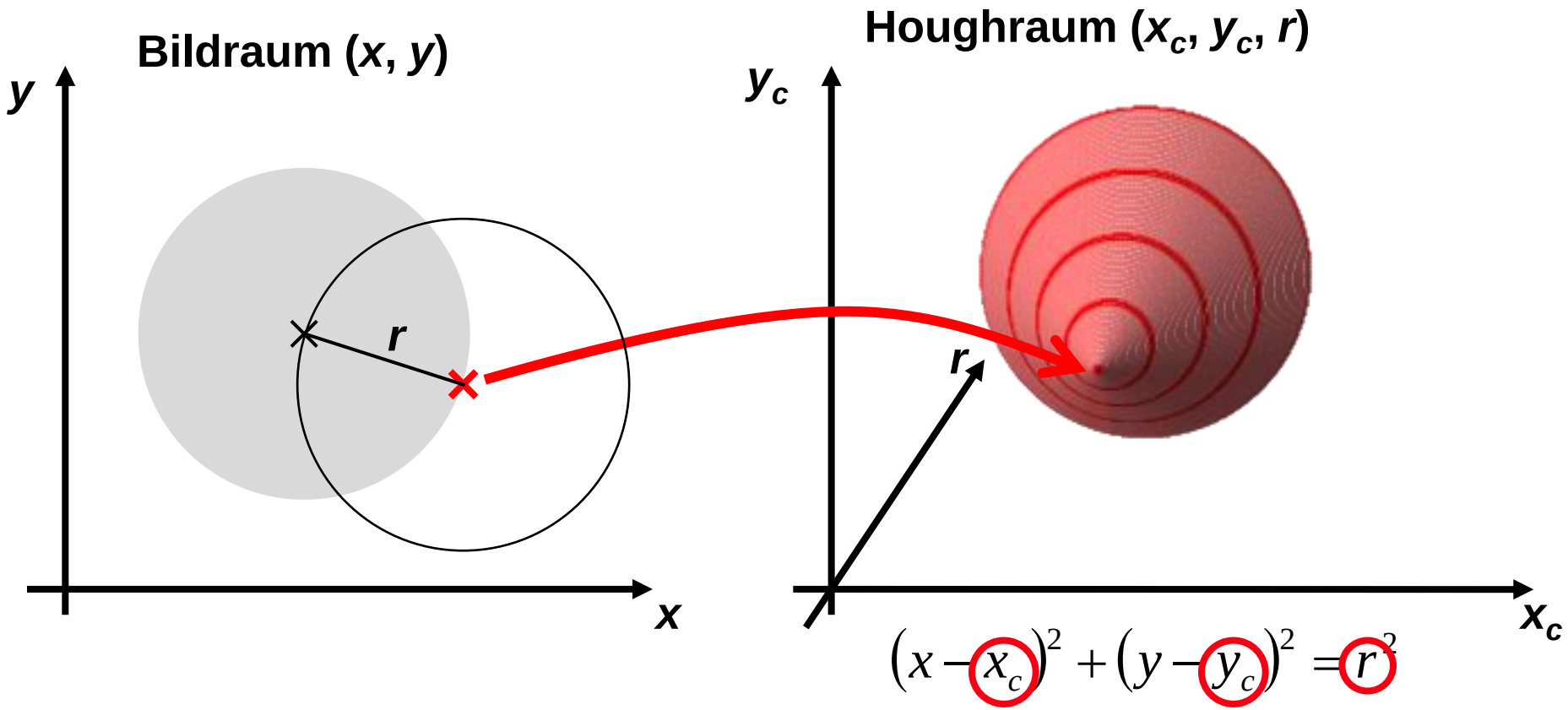


Grundidee der Hough-Transform.

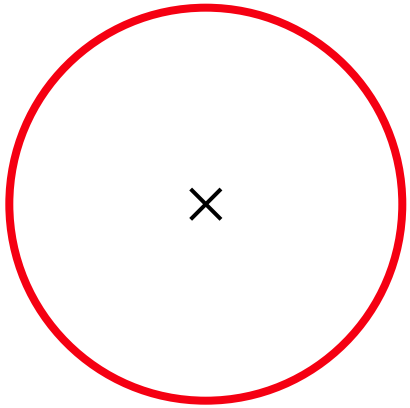


- Hough Algorithmus sucht Kurven
  - in bekannter parametrisierter Form
  - mit unbekannten Parametern
  - für Kreise: Mittelpunkt und Radius
- Eine Form von Merkmalsextraktion
  - Aber: meist zu wenig reichhaltig für ML-Klassifizierung
  - Eher: Man weiß schon was es ist und sucht wo es ist
- Akkumuliert 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 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

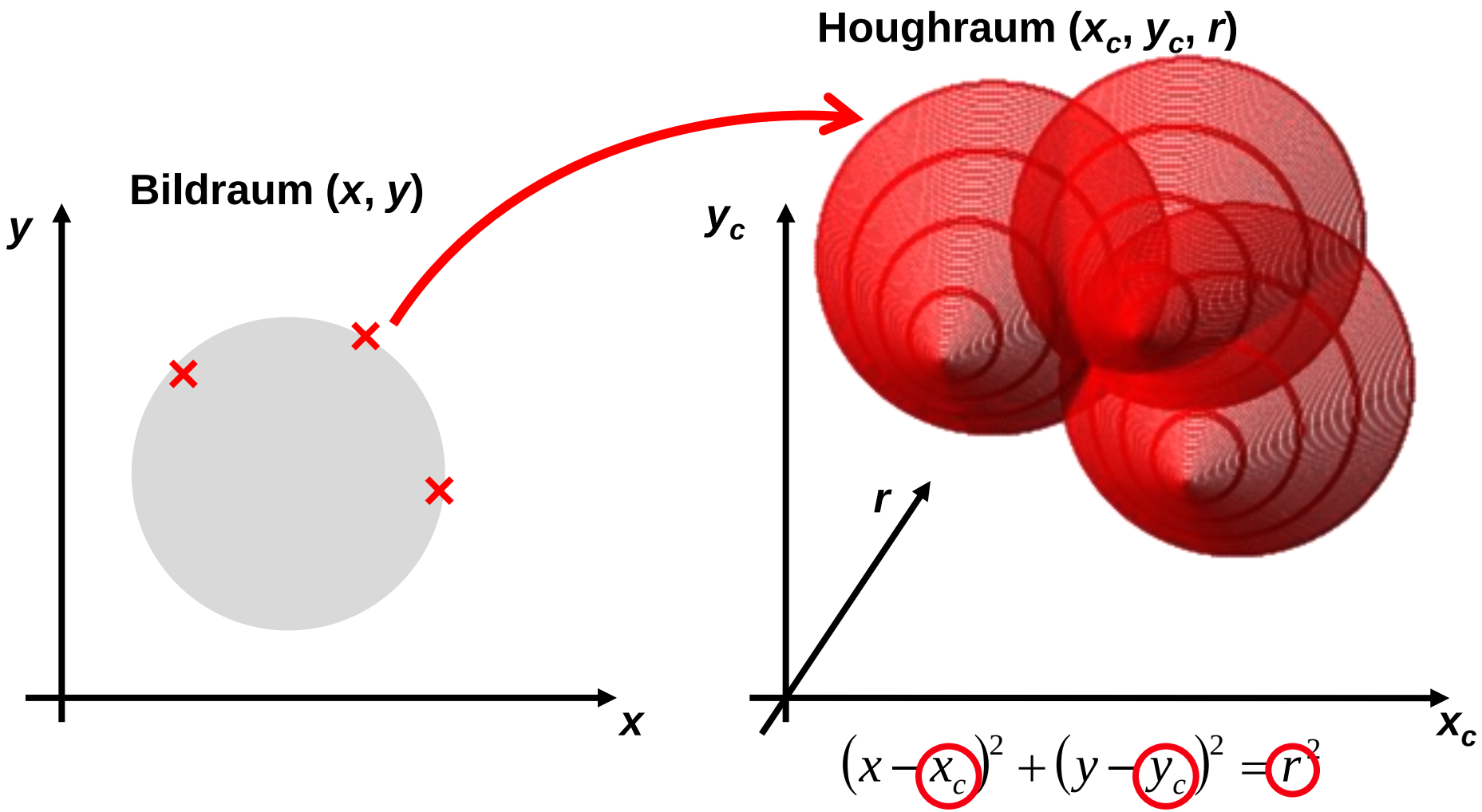
... Evidenz für alle Kurven auf denen er liegt

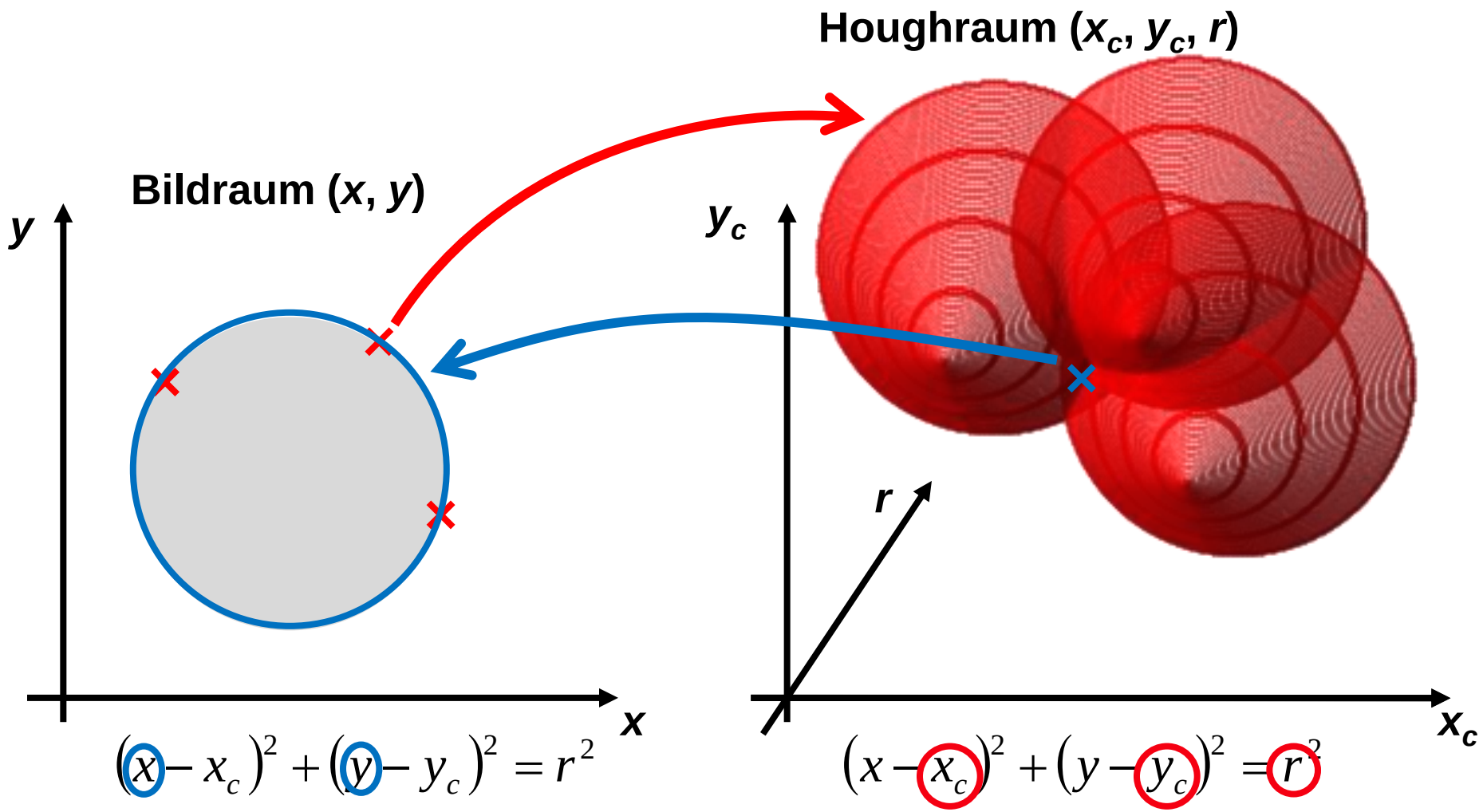


Grundidee der Hough-Transform.



- Hough Algorithmus sucht Kurven
  - in bekannter parametrisierter Form
  - mit unbekannten Parametern
  - für Kreise: Mittelpunkt und Radius
- Eine Form von Merkmalsextraktion
  - Aber: meist zu wenig reichhaltig für ML-Klassifizierung
  - Eher: Man weiß schon was es ist und sucht wo es ist
- 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 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


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


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

Implementierung

- Struktur des Houghraums (x_c, y_c, r)
  - `houghImg [r, yC, xC]`
  - 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 $r \in [r_{min}, r_{max}]$

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

- Akkumulation
  - laufe durch (x_c, y_c)
  - rechne r als aus
  - erhöhe `houghImg[r, yC, xC]` um 1

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

- Problem: 3D Houghraum kostet enormen Speicher und Rechenzeit

Implementierung

- Python-Pseudo-Code ohne Randbehandlung (vgl. Anhang)

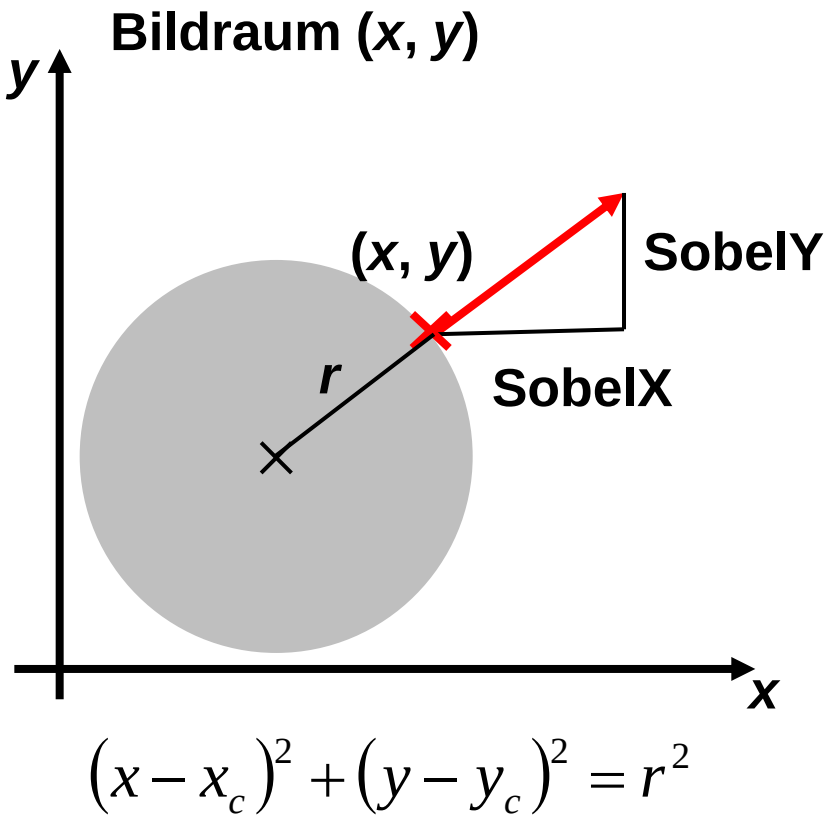
```
def addPointToCHA (houghImg, x, y):
    'increments houghImg at all that pass through (x,y)'
    for dy in range(-RMAX, RMAX+1):
        for dx in range (-RMAX, RMAX+1):
            r = math.hypot (dx, dy) # sqrt(dx²+dy²)
            if r<=RMAX: houghImg [r, y+dY, x+dX] += 1

def circleHough (sobelImg):
    'Takes a gradient image and computes a 3D (r,y,x) circle hough image'
    houghImg = np.zeros ((RMAX, sobelImg.shape[0], sobelImg.shape[1]))
    for y,x in numpy.ndindex (sobelImg.shape[:2]):
          sobelX, sobelY = sobelImg[y,x]
          sobelLen = math.hypot (sobelX, sobelY);
          if sobelLen>sobelThreshold:
             addPointToCHA (houghImg, x, y);
```

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

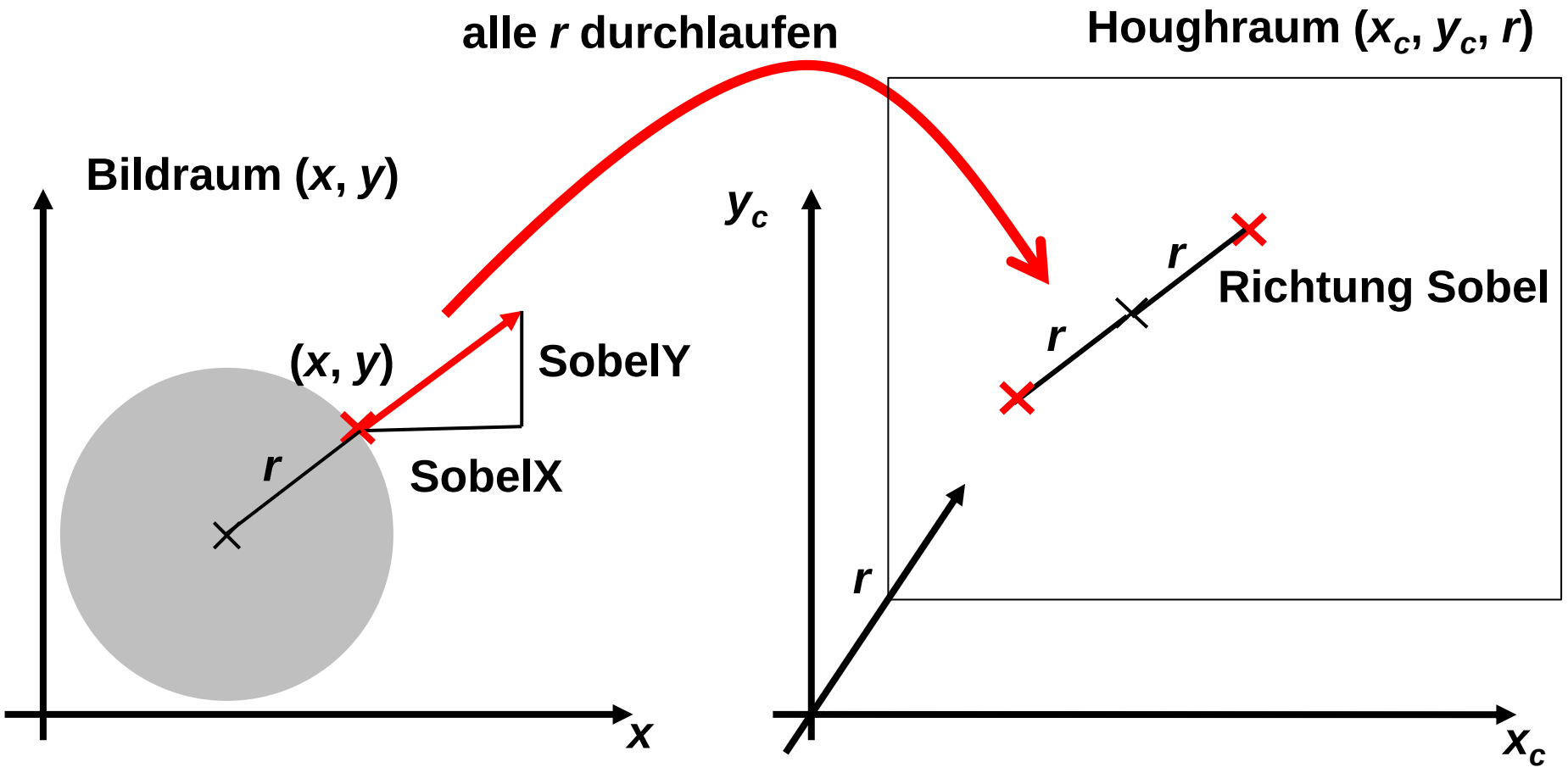
Ausnutzen der Sobelrichtung

- Bei idealem Kreis: Kante tangential
- $\Rightarrow$ Sobel-Vektor radial
- $\Rightarrow$ (SobelX, SobelY) zeigt zum Mittel-punkt oder von ihm weg.
- Idee: Zähle nur noch Kreise, die (x,y) schneiden und dort Radiusrichtung (SobelX, SobelY) haben
  - aussagekräftiger, verbessert Erkennung
  - effizienter, weil weniger potentielle Kreise im Hough-Akkumulator erhöht
- erfüllt (r,y_c,x_c) das Kriterium, liegt (x_c,y_c) von (x,y) um +/-r in Richtung des Sobelvektors



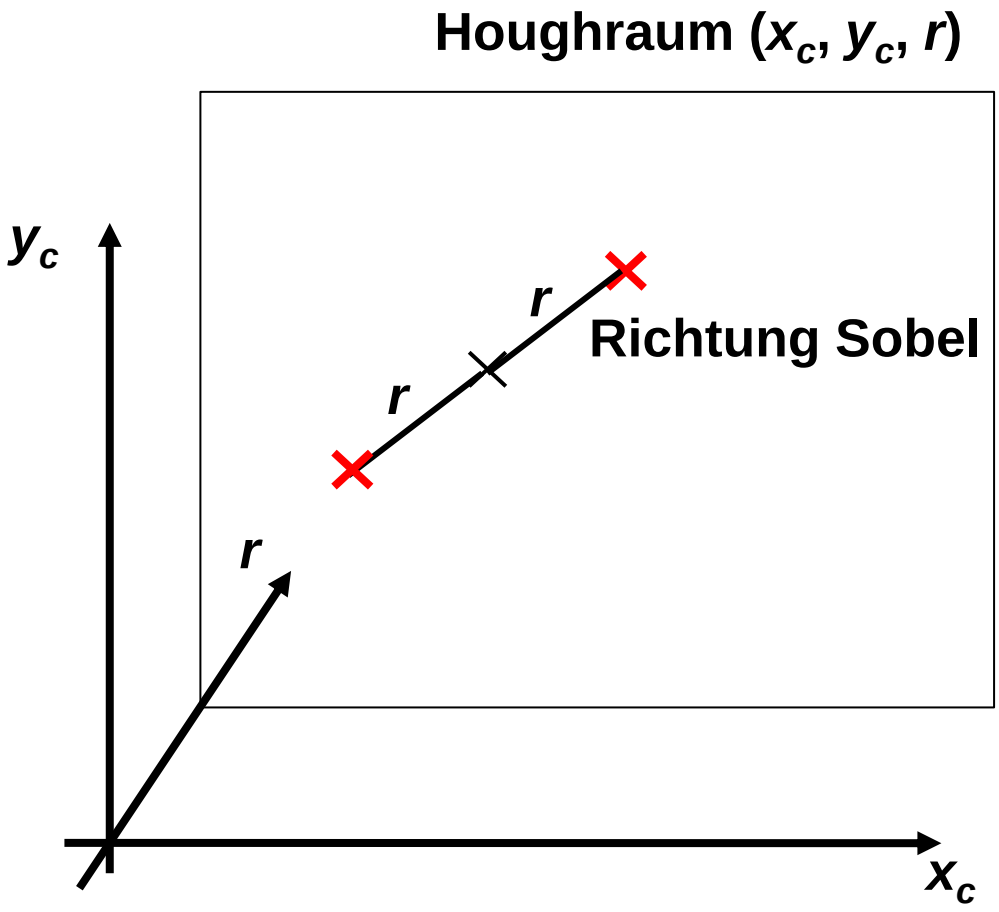
$$(x-x_c)^2+(y-y_c)^2=r^2$$

... Evidenz für alle Kurven ...



... Evidenz für alle Kurven ...

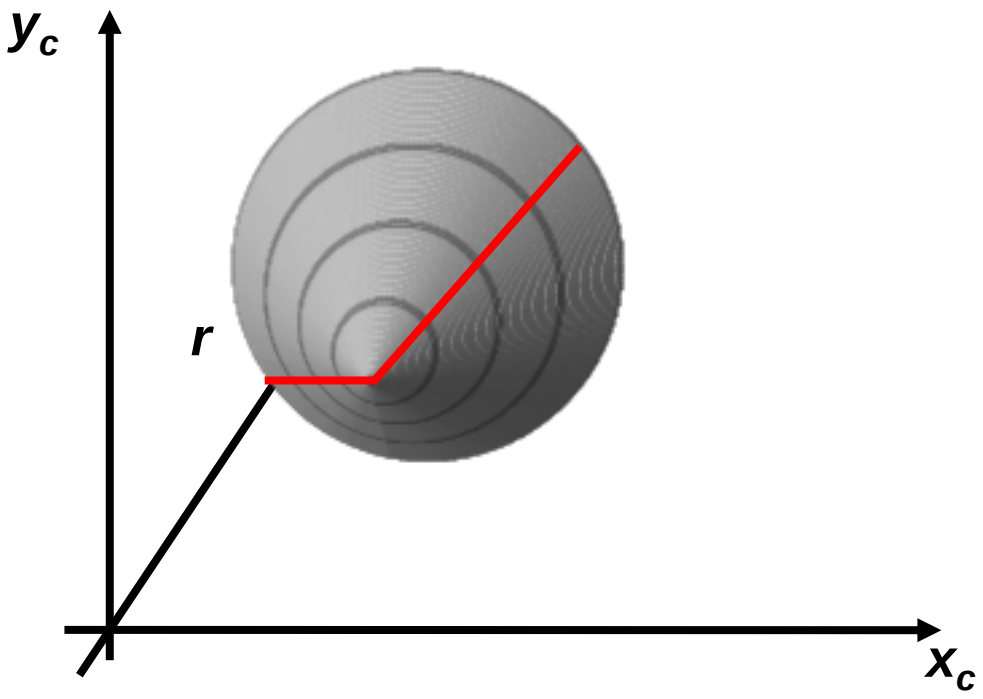
- ... durch (x, y) mit Radius-richtung (*sobelX*, *sobelY*)
- entlang dem Sobelvektors
  r Pixel laufen $r \in [r_{min} \ldots r_{max}]$
  - $x_c = x + r$ sobelX/sobelLen
    $y_c = y + r$ sobelY/sobelLen
  - `houghImg[`r, y_c, x_c`]` erhöhen
- dasselbe entgegen dem Sobelvektor
  - $x_c = x - r$ sobelX/sobelLen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



... Evidenz für alle Kurven ...

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

Houghraum (x_c, y_c, r)



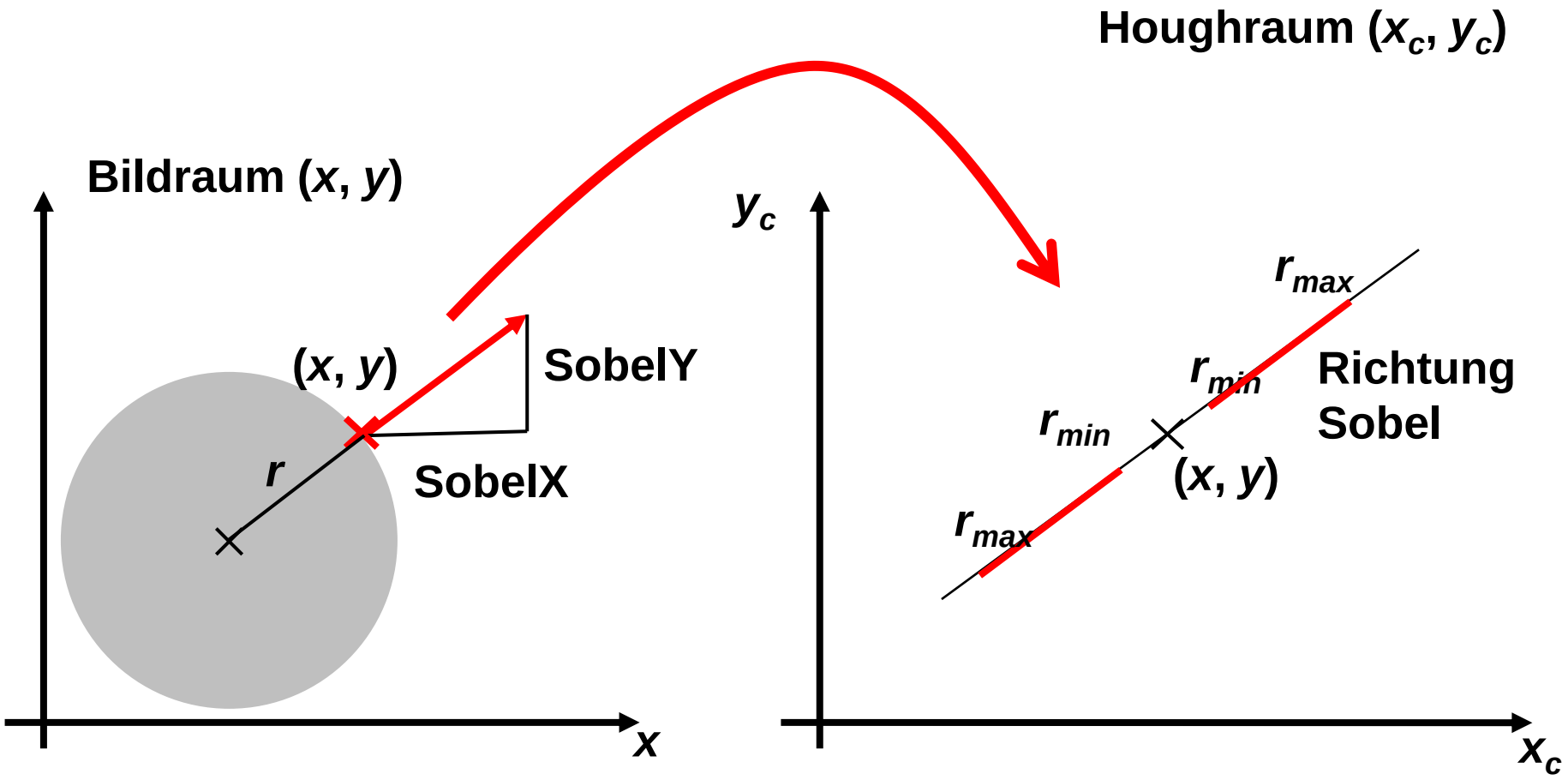
Hough-Transf. Kreise mit Sobelr.

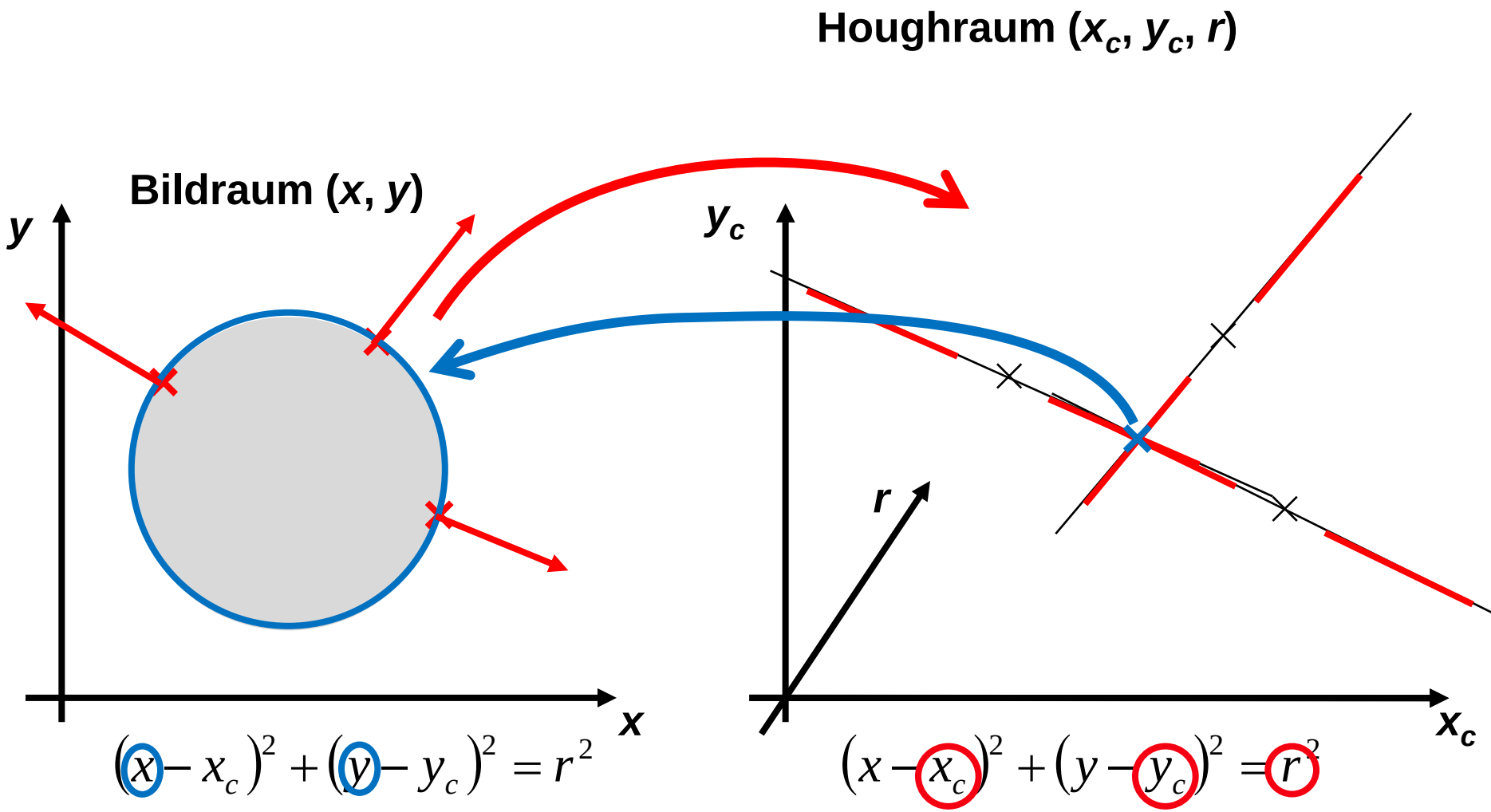
```
def addPointToCHA (houghImg, x, y, sobelX, sobelY):
    sobelLen = math.hypot (sobelX, sobelY)
    for r in range (RMIN, RMAX+1):
        # Entlang Sobelvektor
        xc = x + r*(sobelX/sobelLen);
        yc = y + r*(sobelY/sobelLen);
        houghImg [r, yc, xc] += 1;
        # Entgegen Sobelvektor
        xc = x - r*(sobelX/sobelLen);
        yc = y - r*(sobelY/sobelLen);
        houghImg [r, yc, xc] += 1;
```

Houghraum auf 2D reduzieren

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

... Evidenz für alle Kurven ...



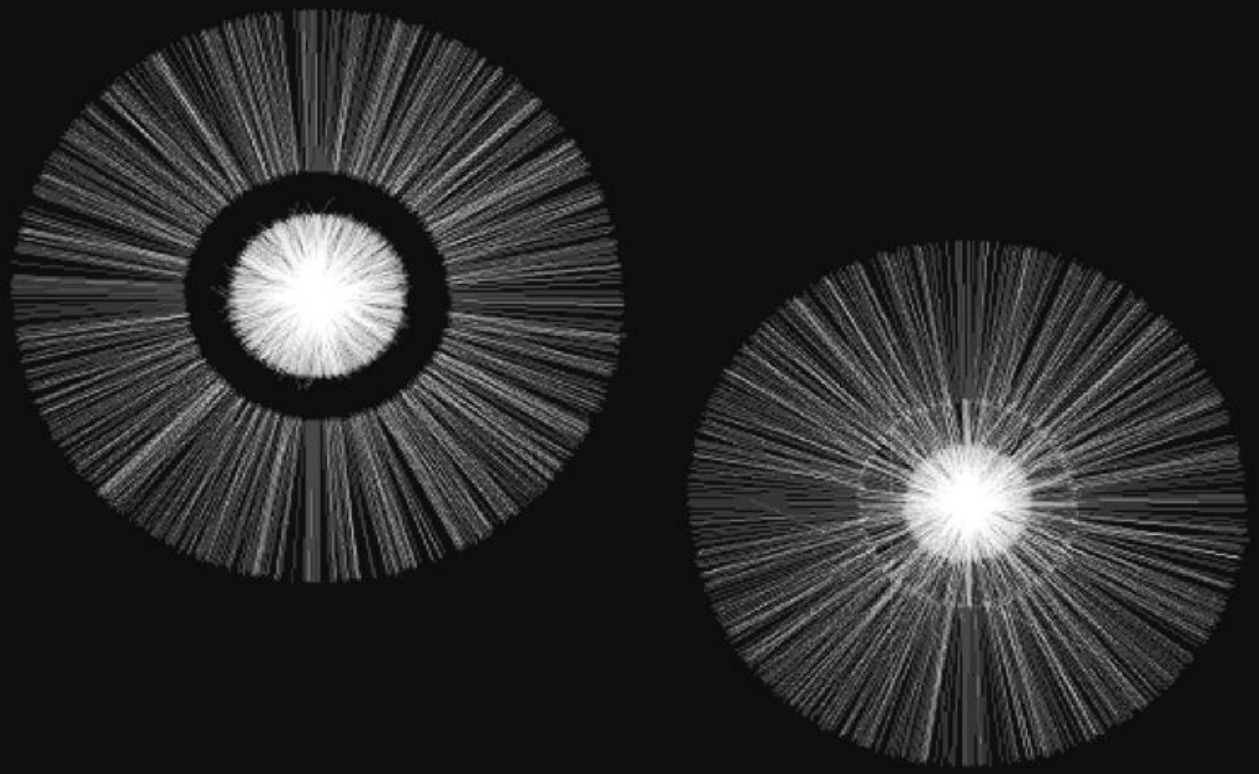

Houghraum (x_c, y_c, r)
Bildraum (x, y)
y
x
y_c
x_c
r
$(x-x_c)^2+(y-y_c)^2=r^2$
$(x-x_c)^2+(y-y_c)^2=r^2$

Testbild: Eingabebild und erkannte Kreise (**rot**)

Testbild: Sobellänge

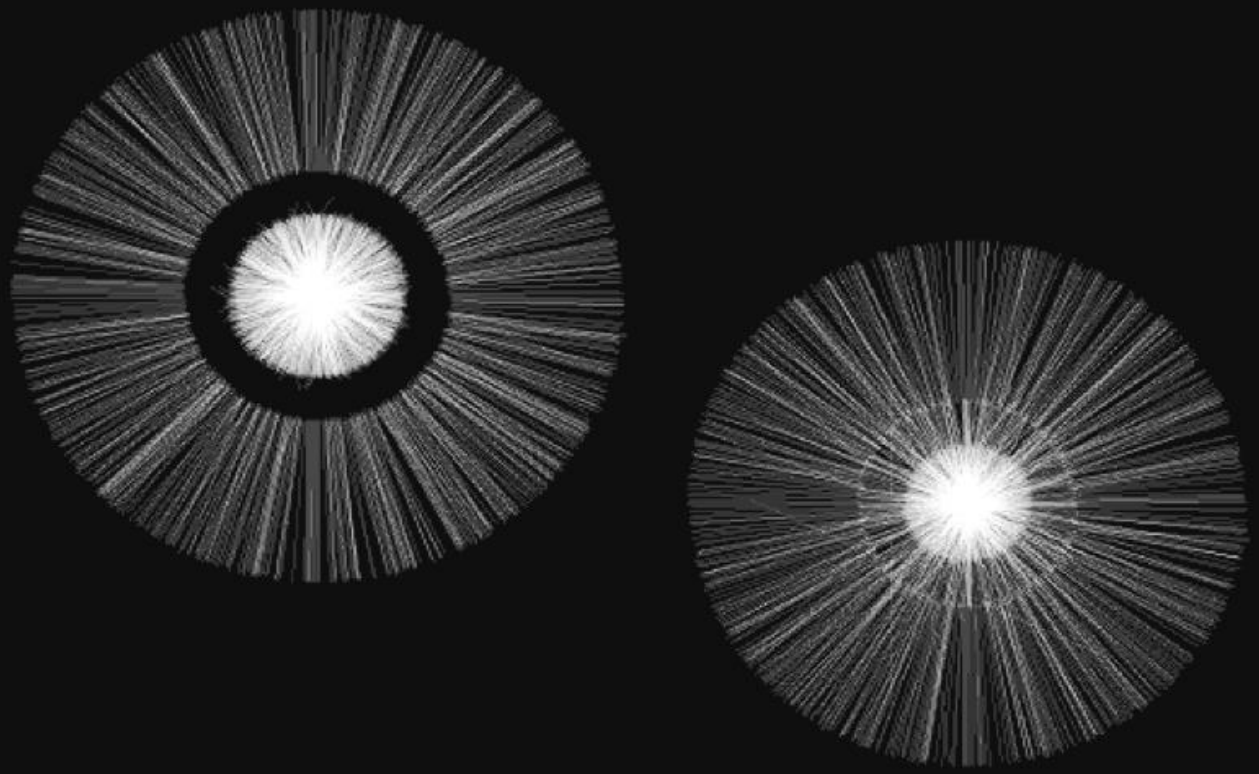
Testbild: Houghraum (aufgehellt)

Frage an das Auditorium: Könnt Ihr das Bild erklären?



Testbild: Houghraum (aufgehellt)

Frage an das Auditorium: Könnt Ihr das Bild erklären?



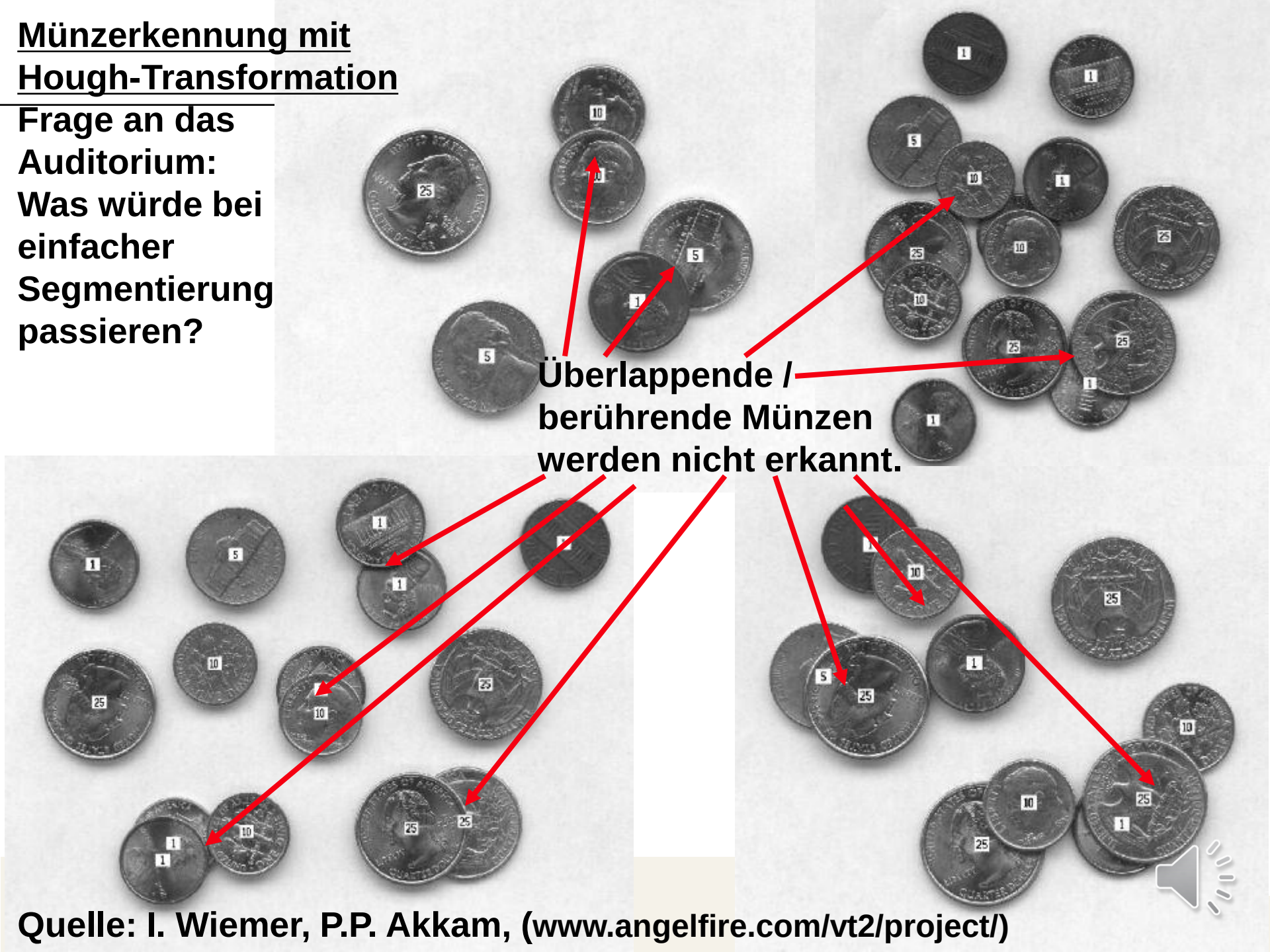
Münzerkennung mit Hough-Transformation

Frage an das Auditorium:
Was würde bei einfacher Segmentierung passieren?

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

Münzerkennung mit Hough-Transformation

Frage an das Auditorium: Was würde bei einfacher Segmentierung passieren?



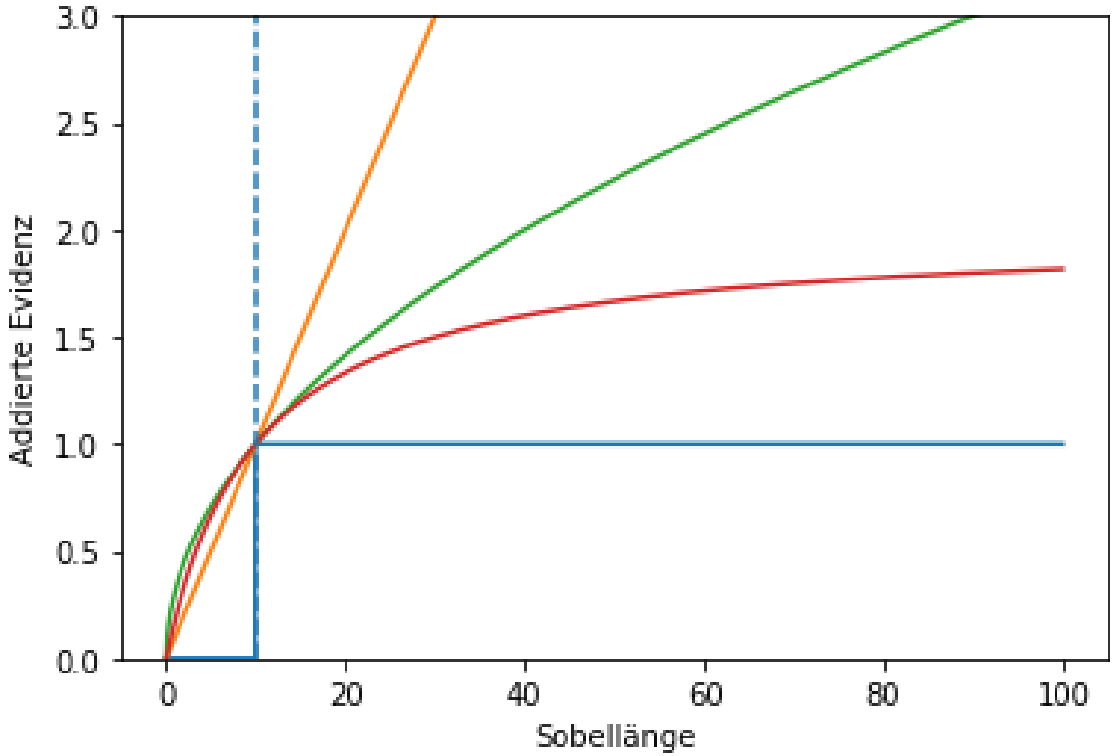
Quelle: I. Wiemer, P.P. Akkam, (www.angelfire.com/vt2/project/)

Keine frühen harten Entscheidungen

- Akkumulation der Evidenzen
  - mustergültiges Beispiel für Entscheidung nach hinten verschieben
  - ggf. Ergebniskreise mit Evidenz herausgeben (Graduelle Ausgaben)
- Schwellwert für Sobellänge
  - effektiv binäres Kantenbild
  - üblich (Literatur und OpenCV)
  - frühe harte Entscheidung

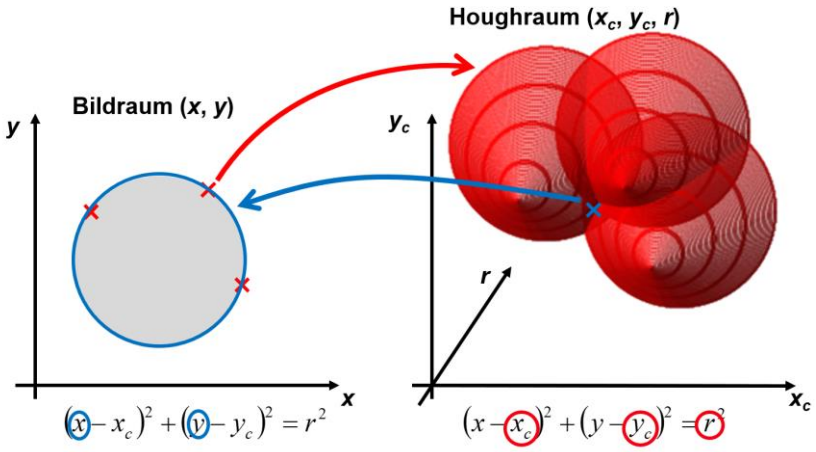
Keine frühen harten Entscheidungen

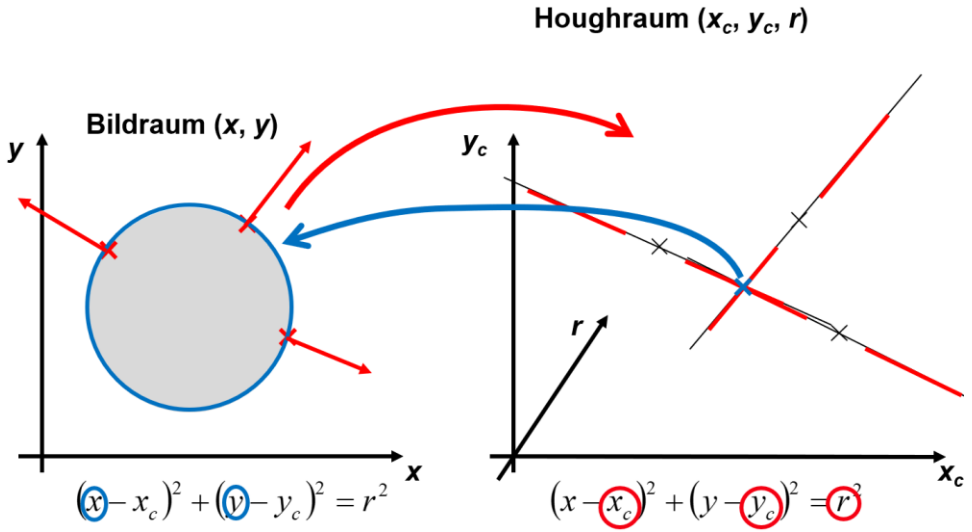
- Aber “Referenzwert“ s^{length}_{ref} nötig (blau gestrichelt)
  - Sobellänge typischer Kante
- Mögliche Funktionen, für die Evidenz in Abhängigkeit der Sobellänge
  - 1 wenn $s^{length} > s^{length}_{ref}$, sonst 0 (**blau**, üblich)
  - $s^{length} / s^{length}_{ref}$ (**orange**),
    → sehr intensive Kante dominiert Ergebnis
  - $\sqrt{s^{length} / s^{length}_{ref}}$ (**grün**)
  - $2s^{length} / s^{length} + s^{length}_{ref}$ (**rot**),
    höchstens 2, empfohlen



Zusammenfassung

- Hough-Transformation
  - Suche parametrisierte Kurven (hier: Kreise)
  - Hough Akkumulator im Parameterraum (hier: x_c, y_c, r)
  - Für jeden Kantenpixel erhöhe alle Akkumulator-Einträge, deren Kurven diesen Pixel durchlaufen
  - Verbesserung: nur Kurven, deren Tangente senkrecht zum Sobel-Vektor ist
  - Optimierung hier: Hough Akkumulator nur Kreismittelpunkte (x_c, y_c)





Python ähnlicher Pseudocode

- Zur Erklärung wie ein Algorithmus funktioniert
  - Lässt oft "sekundäre Fragen" weg
  - z.B. Randtests (`houghImg` Zugriff in `addPointToCHA`)
  - nutzt tlw. mathematische Formelschreibweise
- Würde man nur im Notfall in Python implementieren
  - In Python durch alle Pixel laufen ist mehrere 100mal langsamer als in C
  - vgl. doppelte `for`-Schleife in `circleHough`
- Wo ist Python gut (d.h. auch halbwegs effizient) eingesetzt?
  - "Kommandieren" großer Rechnungen die in C implementiert sind
  - z.B. via numpy oder OpenCv auf Bildern arbeiten
  - Durchführen kleinerer Rechnungen die nur Bruchteil der Rechenzeit sind
  - z.B. Anwendungslogik, Filtern erkannter Objekte