Robotika in računalniško zaznavanje (RRZ)

Detekcija enostavnih krivulj

Danijel Skočaj

Univerza v Ljubljani

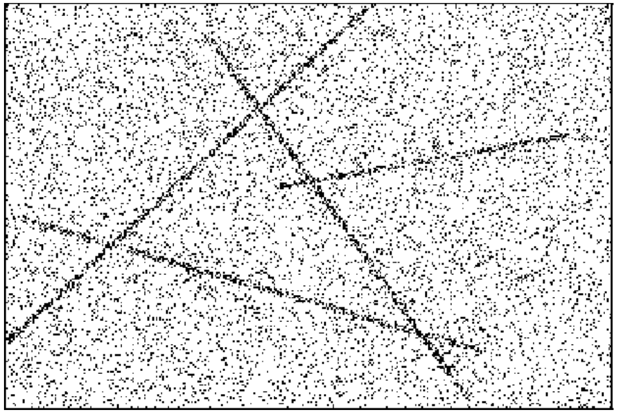
Fakulteta za računalništvo in informatiko

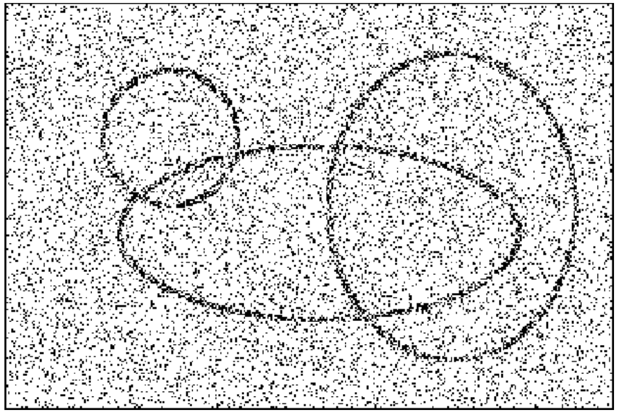
Literatura: W. Burger, M. J. Burge (2008).
Digital Image Processing, poglavje 9

v1.0

Enostavne krivulje

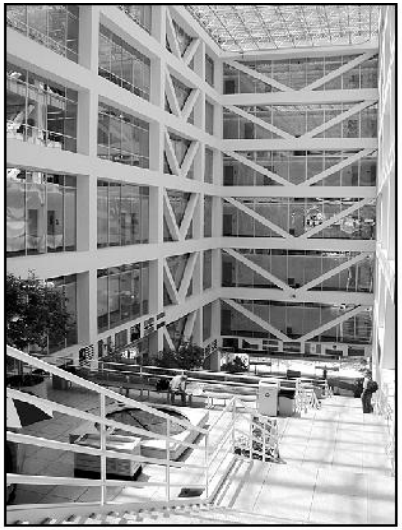
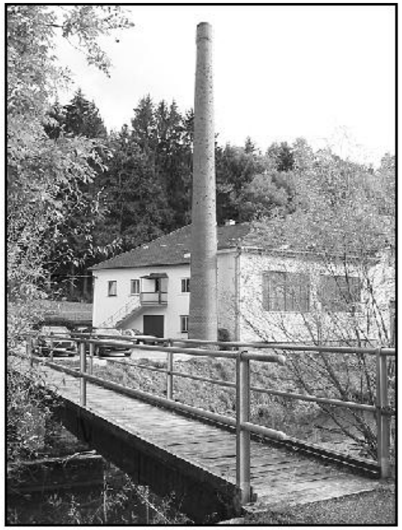
- Rezultat detektorjev robov je binarna slika robov
  - Vsebuje tudi veliko šuma (napačno detektiranih robnih slikovnih elementov)
- Želimo robne slikovne elemente povezati med sabo, detektirati višje strukture – enostavne krivulje (premice, krožnice, elipse,...)
- Dva pristopa
  - Od spodaj navzgor – sledenje sosednjim robnim sl. elementom
  - Od zgoraj navzdol (globalno) – iskanje določenih modelov na globalni sliki; Houghova transformacija



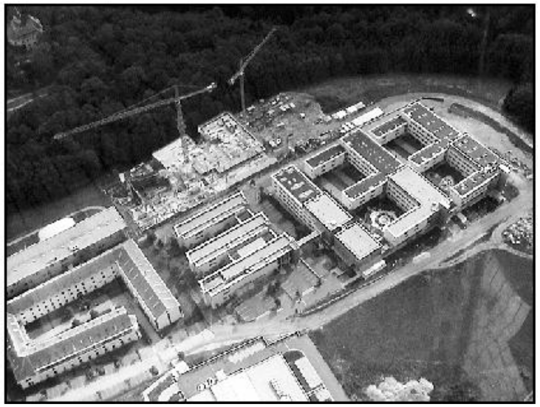


Enostavne geometrične oblike

- Črte, krogi, elipse
- Zelo pogoste v umetno ustvarjenih okoljih



Houghova transformacija

- Globalna metoda za detekcijo premic, krožnic, drugih parametričnih krivulj
- Najpogosteje uporabljana za detekcijo premic

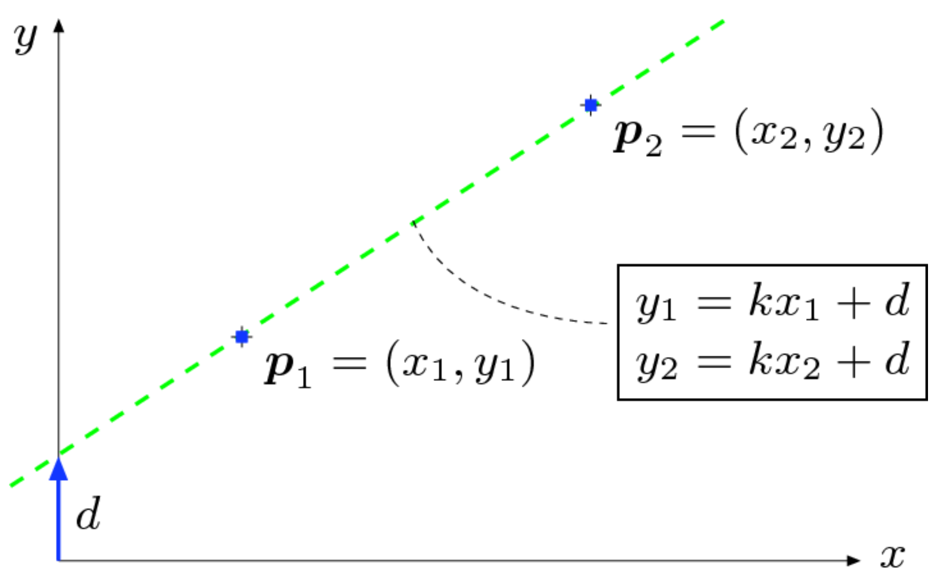
- Na sliki je premica, če na njej leži veliko število robnih slikovnih elementov
- Naivna ideja: zgeneriraj vse možne premice in preveri koliko robnih slikovnih elementov leži na vsaki od njih.
  - Za vsako premico preveri vse slikovne elemente
  - Preveč potratno!
- Boljša ideja: Za vsak slikovni element poglej na katerih premicah bi lahko ležal
  - Delamo v prostoru parametrov premic
  - Precej manj potratno

Predstavitev premice

- Predstavitev premice v 2D:

$$y = kx + d$$

- Za točki $\boldsymbol{p}_1 = (x_1, y_1)$ and $\boldsymbol{p}_2 = (x_2, y_2)$
  ki ležita na premici, velja $y_1 = kx_1 + d$ and $y_2 = kx_2 + d$
- Poiskati moramo vrednosti k in d tako, da bo čimveč točk (parov x_i, y_i) ležalo na premici

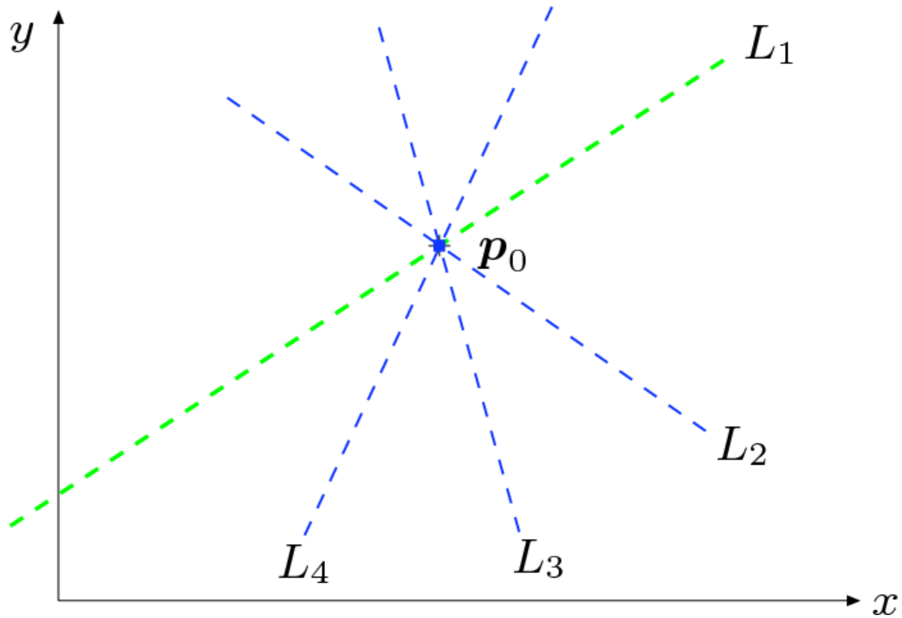


Parametrični oz. Houghov prostor

- Preglejmo vse možne premice, ki lahko gredo skozi vsako dano točko na sliki
  - Skozi točko p_0 :

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

$$d_j = -x_0 k_j + y_0$$



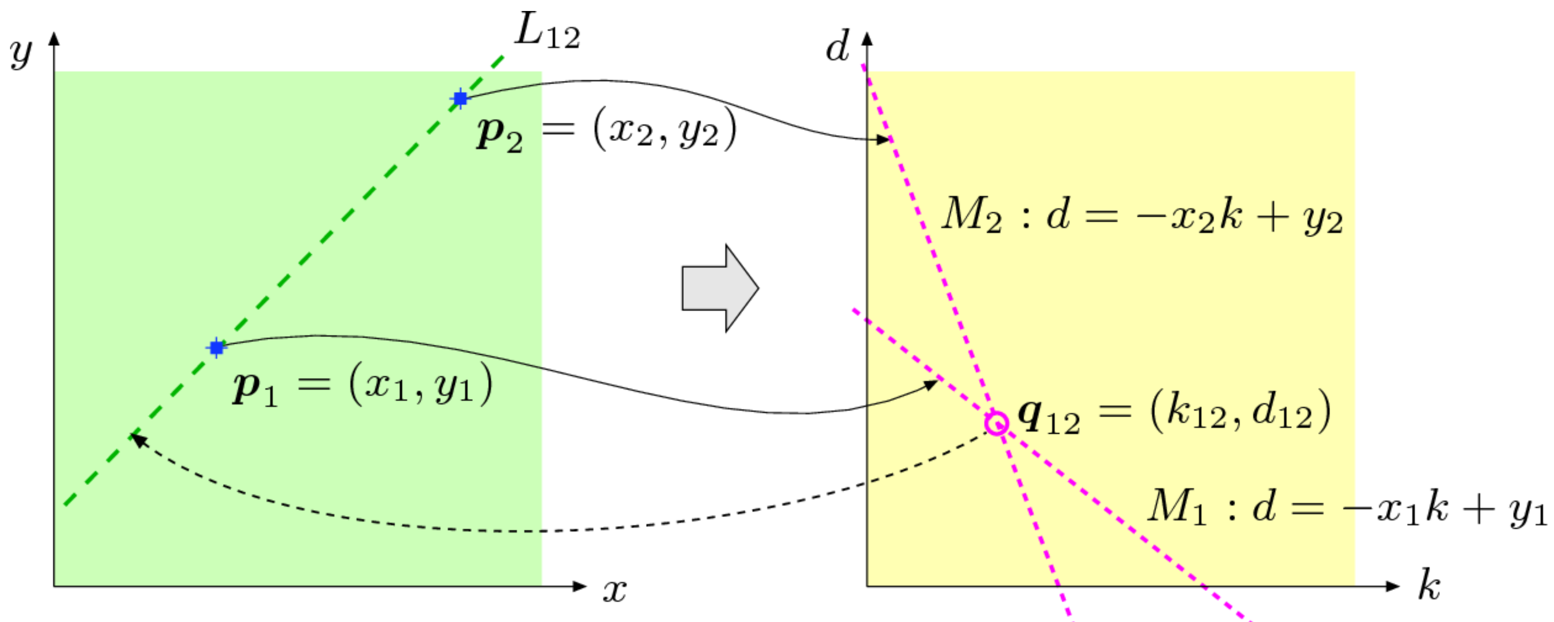
  - Skozi poljubno točko p_i :

$$M_i : d = -x_i k + y_i$$

    - Xi in yi sta parametra parametričnem prostoru, ki ga napenjata k in d!

Prostora slik in parametrov

Image Space (x, y)		*Parameter Space* (k, d)	
Point	$\boldsymbol{p}_i = (x_i, y_i)$	$M_i : d = -x_i k + y_i$	Line
Line	$L_j : y = k_j x + d_j$	$\boldsymbol{q}_j = (k_j, d_j)$	Point



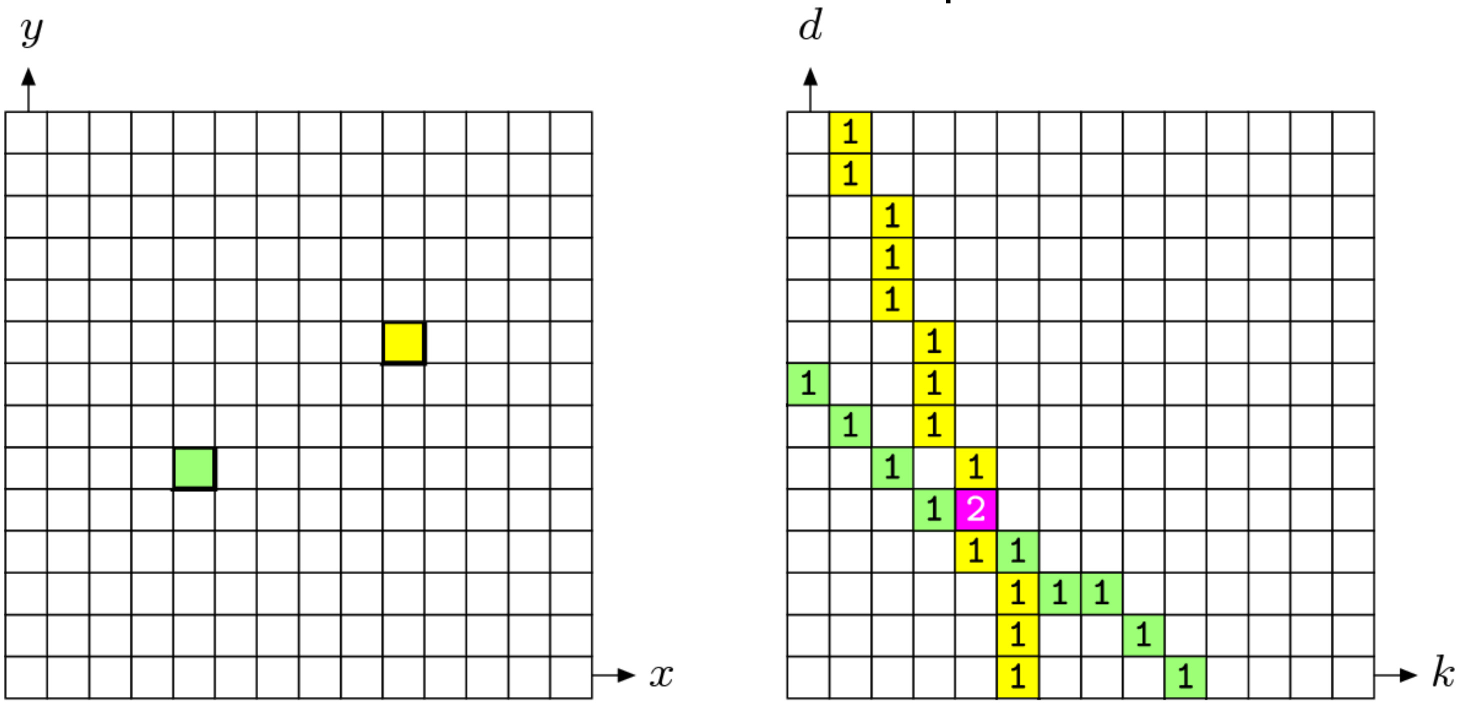
(a) x/y image space

(b) k/d parameter space

- Presečišče *N* premic v prostoru parametrov na lokaciji (k', d') -> v prostoru slik *N* točk leži na premici $y = k'x + d'$

Akumulatrosko polje

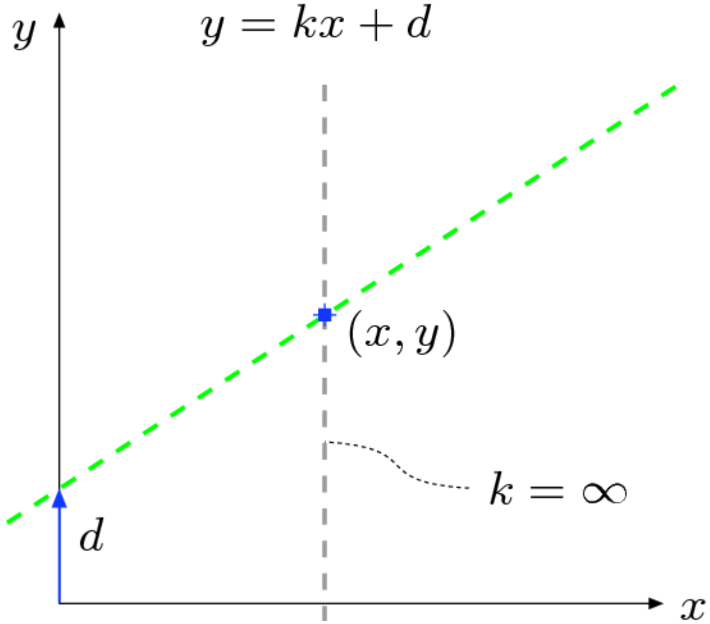
- Iščemo lokacije v param. prostoru, kjer se seče veliko premic
- Diskretiziramo parametični prostor -> akumulatorsko polje
- Vsaka premica v prostoru slik glasuje za eno celico v akumulatorskem polju
  - HT išče maksimume v akumulatorksem prostoru

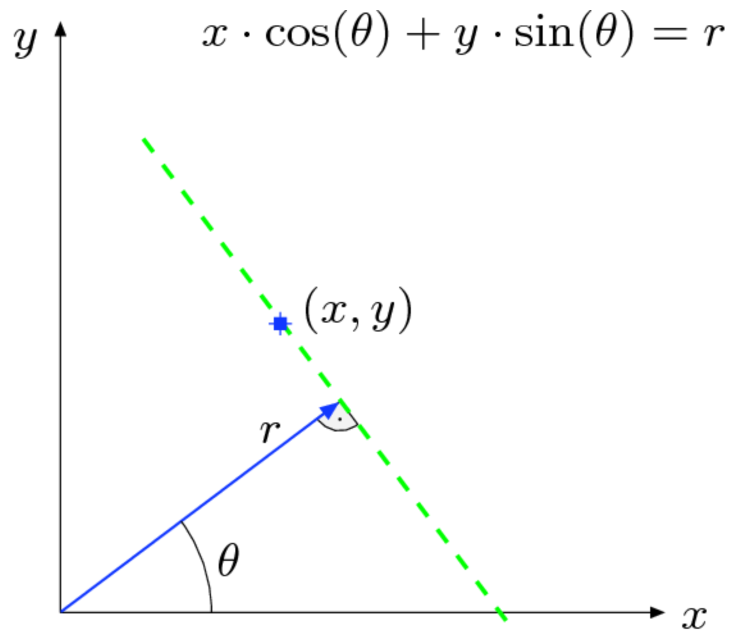


(a) Image Space (b) Accumulator Array

Boljša predstavitev za premico

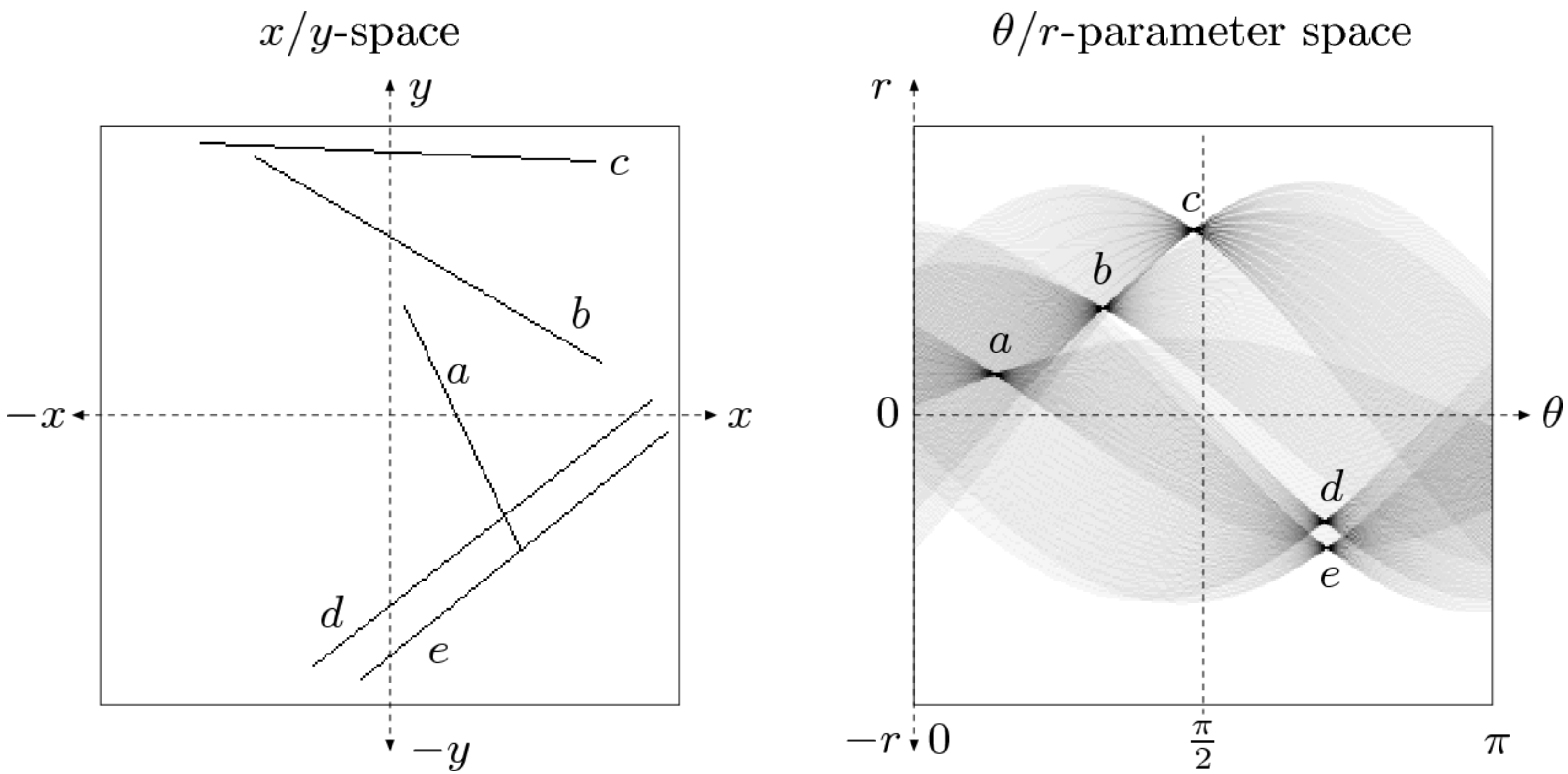
- Standardna predstavitev premice: $y = kx + d$
  - Nelinearno spreminjanje k
  - Problem pri $k = \infty$
- Hessova normalna oblika (HSN): $x \cdot \cos(\theta) + y \cdot \sin(\theta) = r$
  - Parametra r in θ (radij in kot)
  - Linearno spreminjanje obeh parametrov
  - Ni singularnosti





Akumulatorsko polje

- Akumulatrosko polje v HNF
  - Računamo radij: $r_{x_i,y_i}(\theta) = x_i \cdot \cos(\theta) + y_i \cdot \sin(\theta)$
  - Meje: kot: $0 \leq \theta < \Pi$
    radij: $-r_{\max} \leq r_{x,y}(\theta) \leq r_{\max}, \quad \text{where} \quad r_{\max} = \frac{1}{2}\sqrt{M^2 + N^2}$



Algoritem

$\textsc{HoughLines}(I)$

Returns the list of parameters $\langle \theta_i, r_i \rangle$ corresponding to the strongest lines found in the binary image I.

Set up a two-dimensional array $Acc[\theta, r]$ of counters, initialize to 0.

Let (u_c, v_c) be the center coordinates of the image I

for all image coordinates (u, v) **do**

    if $I(u, v)$ is an edge point **then**

        Get coordinate relative to the image center (u_c, v_c):

        $(x, y) \leftarrow (u - u_c, v - v_c)$

        for $\theta_i = 0 \ldots \pi$ **do**

            $r_i = x \cdot \cos(\theta_i) + y \cdot \sin(\theta_i)$

            Increment $Acc[\theta_i, r_i]$

Return the list of parameter pairs $\langle \theta_j, r_j \rangle$ for K strongest lines:

$MaxLines \leftarrow \textsc{FindMaxLines}(Acc, K)$

return $MaxLines$.

Algoritem

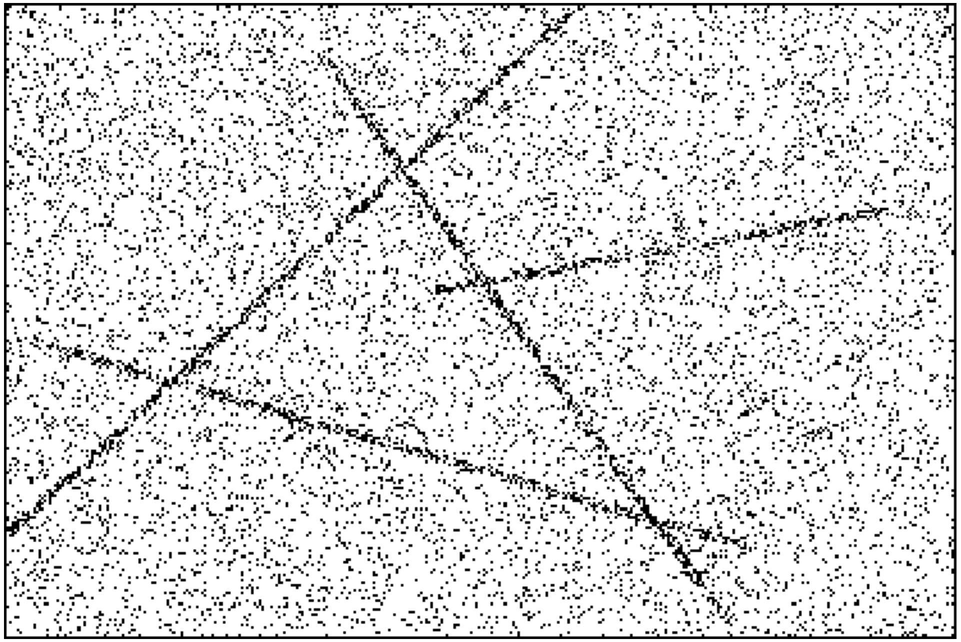
- Potrebno je pravilno diskretizirati akumulatorsko polje

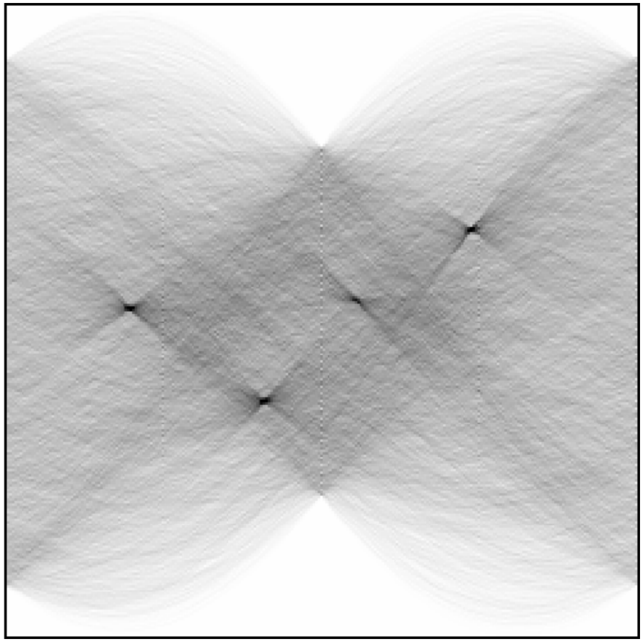
$$\Delta_\theta = \frac{\pi}{N_\theta} \quad \text{and} \quad \Delta_r = \frac{2 \cdot r_{\max}}{N_r}$$

```
class LinearHT {
  ImageProcessor ip; // reference to the original image I
  int xCtr, yCtr;    // x/y-coordinates of image center (u_c, v_c)
  int nAng;          // N_θ steps for the angle (θ = 0...π)
  int nRad;          // N_r steps for the radius (r = -r_max...r_max)
  int cRad;          // center of radius axis (r = 0)
  double dAng;       // increment of angle Δ_θ
  double dRad;       // increment of radius Δ_r
  int[][] houghArray; // Hough accumulator Acc[θ_i, r_i]

  //constructor method:
  LinearHT(ImageProcessor ip, int nAng, int nRad) {
    this.ip = ip;
    this.xCtr = ip.getWidth()/2;
    this.yCtr = ip.getHeight()/2;
    this.nAng = nAng;
    this.dAng = Math.PI / nAng;
    this.nRad = nRad;
    this.cRad = nRad / 2;
    double rMax = Math.sqrt(xCtr * xCtr + yCtr * yCtr);
    this.dRad = (2.0 * rMax) / nRad;
    this.houghArray = new int[nAng][nRad];
    fillHoughAccumulator();
  }

  void fillHoughAccumulator() {
    int h = ip.getHeight();
    int w = ip.getWidth();
    for (int v = 0; v < h; v++) {
      for (int u = 0; u < w; u++) {
        if (ip.get(u, v) > 0) {
          doPixel(u, v);
        }
      }
    }
  }

  void doPixel(int u, int v) {
    int x = u - xCtr, y = v - yCtr;
    for (int i = 0; i < nAng; i++) {
      double theta = dAng * i;
      int r = cRad + (int) Math.rint
          ((x*Math.cos(theta) + y*Math.sin(theta)) / dRad);
      if (r >= 0 && r < nRad) {
        houghArray[i][r]++;
      }
    }
  }

} // end of class LinearHT
```

Analiza akumulatorskega polja

- Detekcija maksimumov v akumulatorskem polju
  - So „razmazani“, lokalni maksimumi
  - Zaradi šuma, diskretizacije

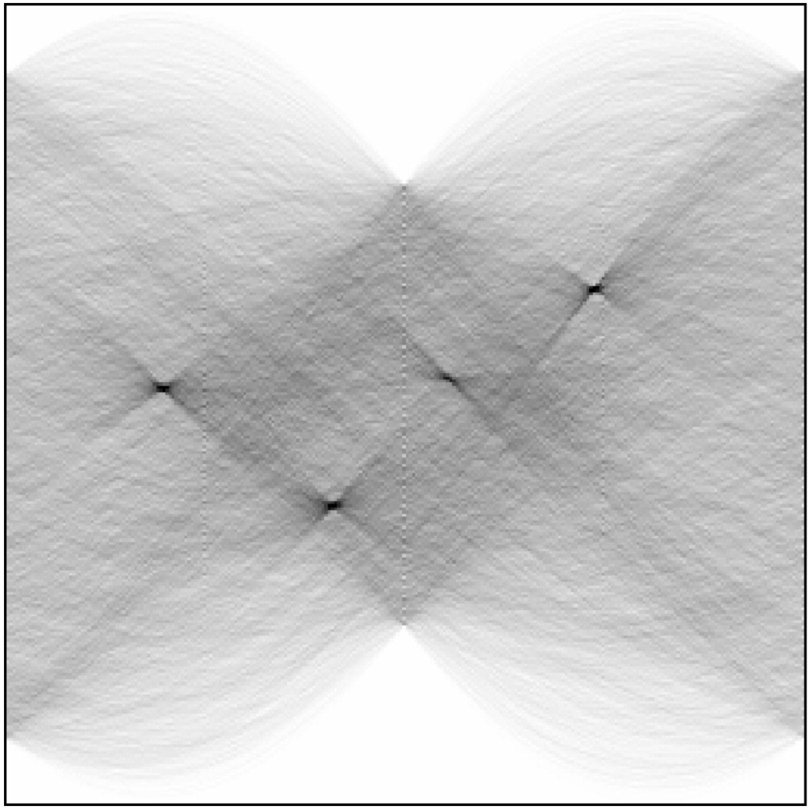


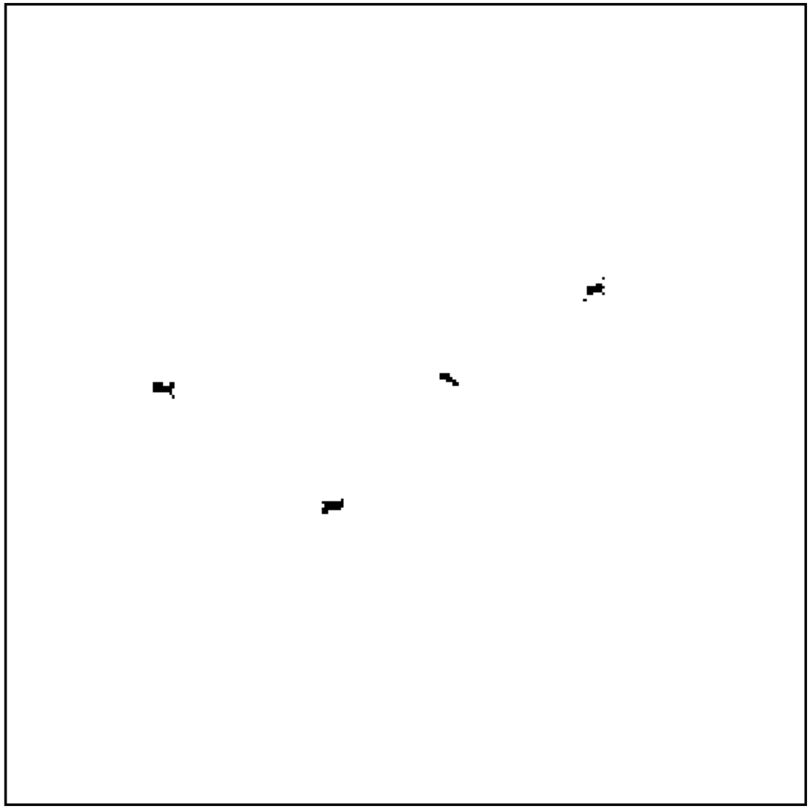


- Dva pristopa:
  - Upragovljenje
  - Dušitev ne-maksimumov

Upragovljenje

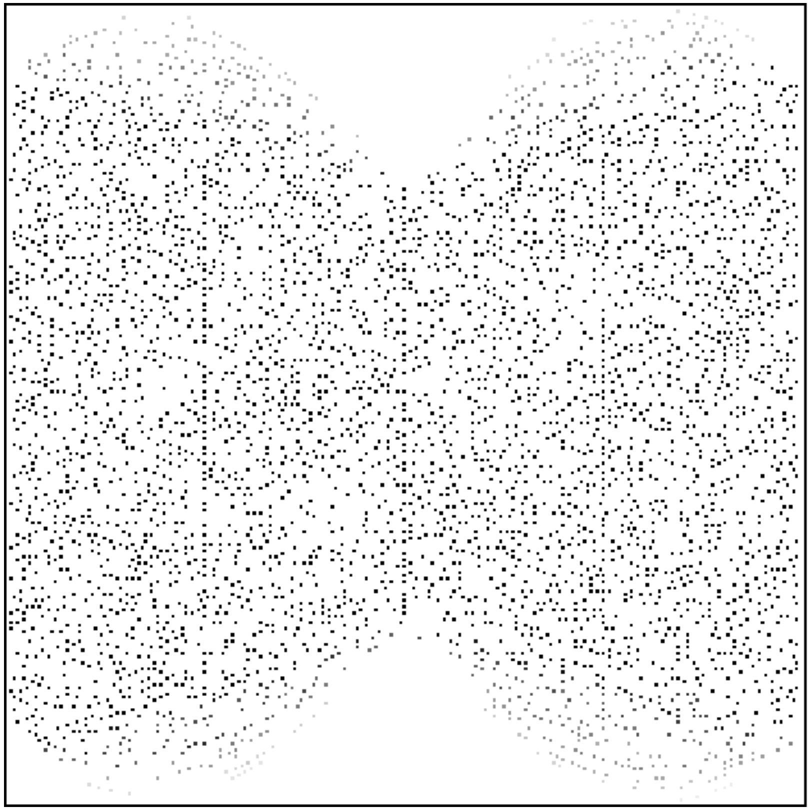
- Upragovi akumulatorsko polje
- Poišči lokalne regije
- Centri regij določajo premice

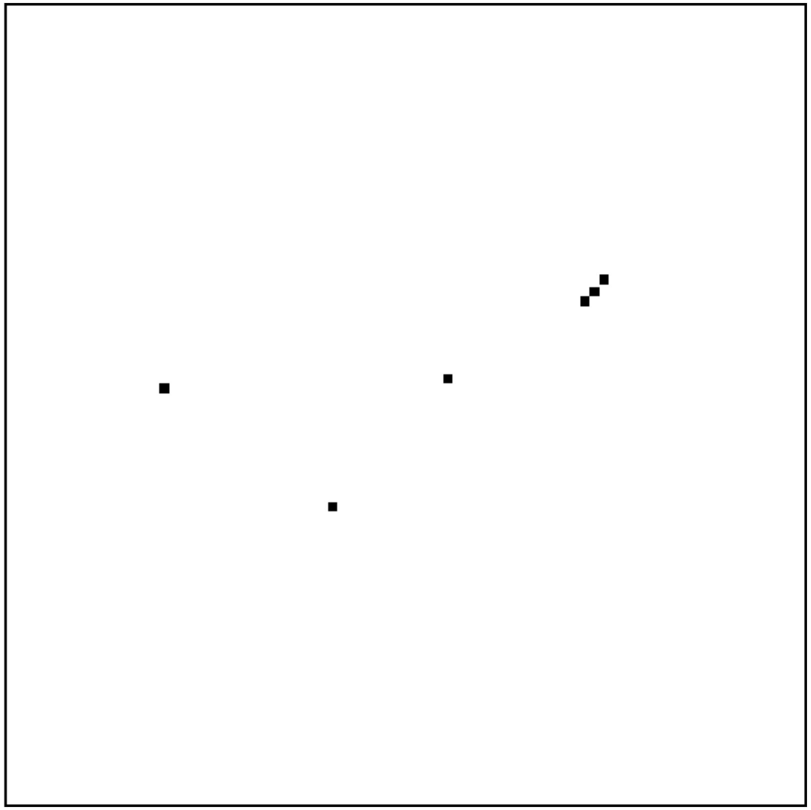




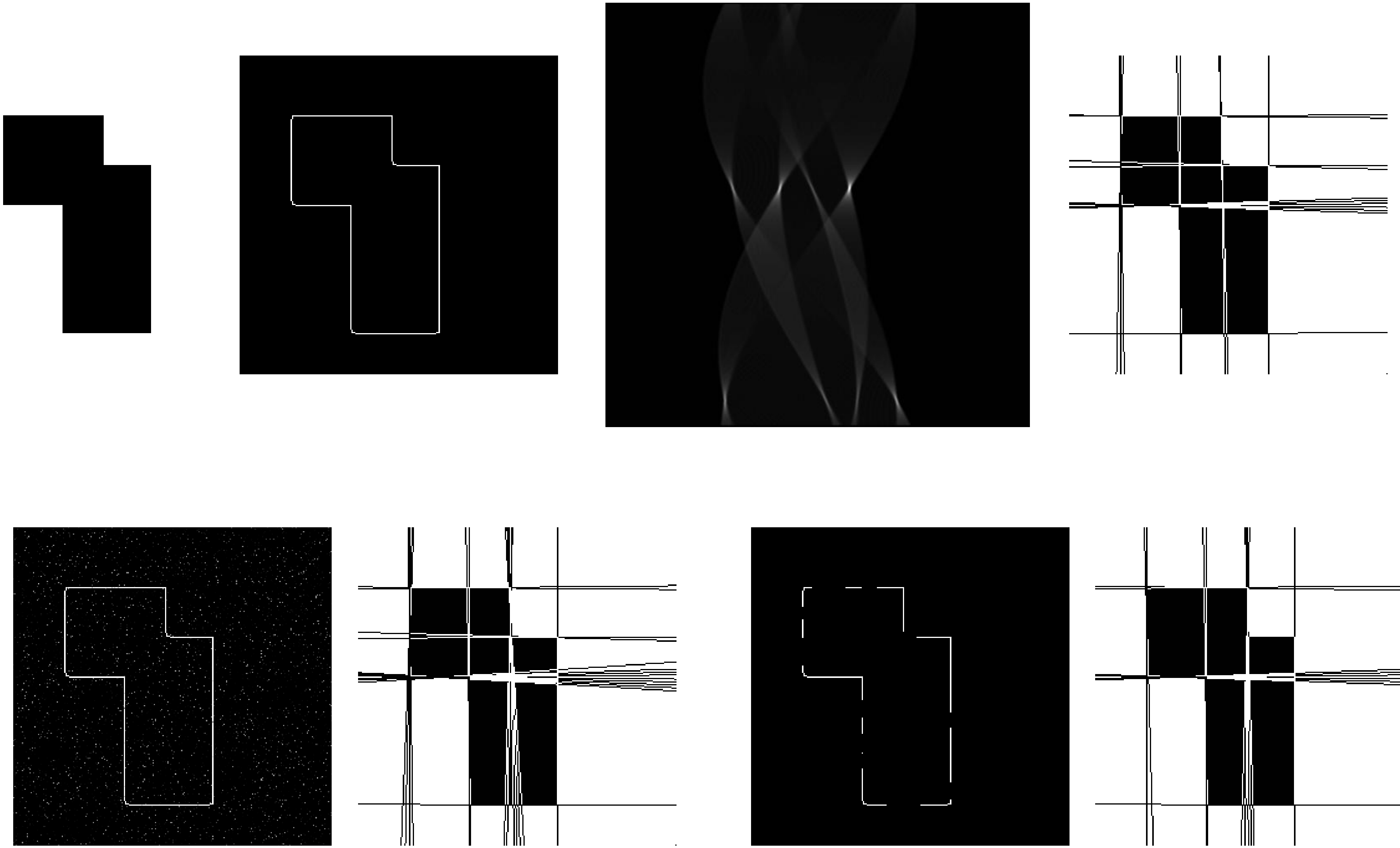
Dušitev ne-maksimumov

- Dovolimo samo en maksimum v okolici
- Lahko kombiniramo z upragovljenjem





Primer

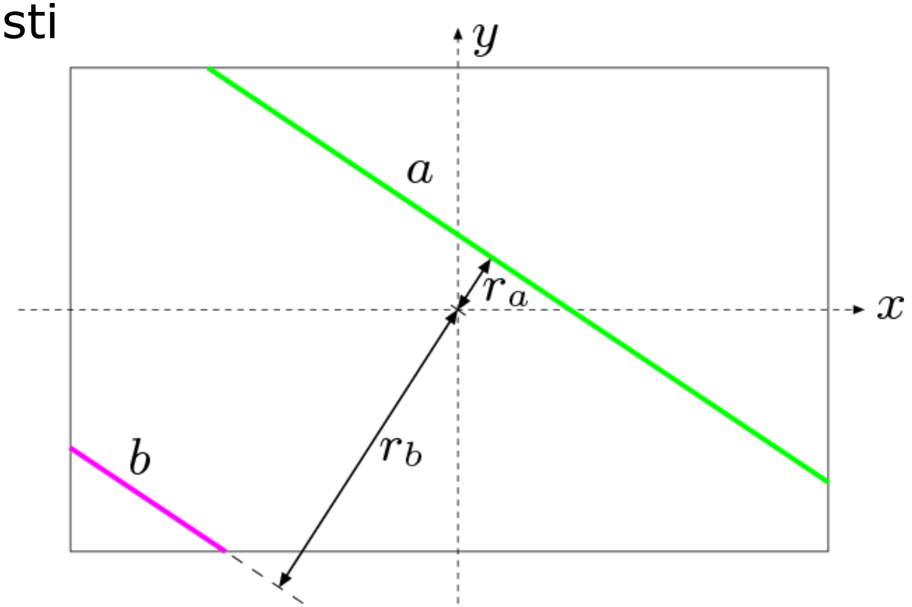


Razširitve

- Izboljšano osveževanje akumulacijskega polja
  - Problem: zaradi diskretizacije prostora parametrov in zaokroževanja, lahko glasovi padejo v sosednjo celico
  - Rešitev: osveži tudi sosednje celice

- Problem kratkih premic
  - Problem: premice v vogalih slik so krajše, na njih leži manj točk, imajo manjšo podporo v akumulacijskem polju:
  - Rešitev: normalizacija vrednosti v akumulatroskem polju:

$$Acc'[\theta, r] \leftarrow \frac{Acc[\theta, r]}{MaxHits[\theta, r]}$$



Razširitve

- Iskanje daljic
  - Osnovna verzija HT detektira premice

  -> Hranimo tudi začetno in končno točko daljice
  - Dva načina:
    - V drugem koraku pogledamo kateri slikovni elementi ležijo na detektirani premici in poiščemo ekstreme
    - Ko gradimo akumulatorsko polje si zapomnimo tudi min. in max. koordinate:

$$Acc[\theta, r] = \langle count, x_{\mathrm{start}}, y_{\mathrm{start}}, x_{\mathrm{end}}, y_{\mathrm{end}} \rangle$$

Razširitve

- Iskanje presečišč premic

  - Presečišče $\boldsymbol{x}_0 = (x_0, y_0)^T$

    premic $L_1 = \langle \theta_1, r_1 \rangle \quad \text{and} \quad L_2 = \langle \theta_2, r_2 \rangle$

    dobimo z rešitvijo sistema linearnih enačb

$$x_0 \cdot \cos(\theta_1) + y_0 \cdot \sin(\theta_1) = r_1$$
$$x_0 \cdot \cos(\theta_2) + y_0 \cdot \sin(\theta_2) = r_2$$

  - Rešitev je:

$$\begin{aligned}
\boldsymbol{x}_0 &= \frac{1}{\cos(\theta_1)\sin(\theta_2) - \cos(\theta_2)\sin(\theta_1)} \cdot \begin{bmatrix} r_1 \sin(\theta_2) - r_2 \sin(\theta_1) \\ r_2 \cos(\theta_1) - r_1 \cos(\theta_2) \end{bmatrix} \\
&= \frac{1}{\sin(\theta_2 - \theta_1)} \cdot \begin{bmatrix} r_1 \sin(\theta_2) - r_2 \sin(\theta_1) \\ r_2 \cos(\theta_1) - r_1 \cos(\theta_2) \end{bmatrix}
\end{aligned}$$

Razširitve

- Upoštevanje moči in smeri roba
  - Uporabljamo sivinsko sliko roba in ne binarno sliko
  - Vrednost akumulatorskega polje povečujemo za vrednosti jakosti roba v danem slikovnem elementu:

$$Acc[\theta, r] \leftarrow Acc[\theta, r] + E(u, v)$$

  - Orientacija roba nam laho bistveno omeji število kotov za katere računamo radij za posamezno premico
    - Zelo pohitrimo računanje HT
    - Zmanjša tudi število vseh glasov v ak. polju

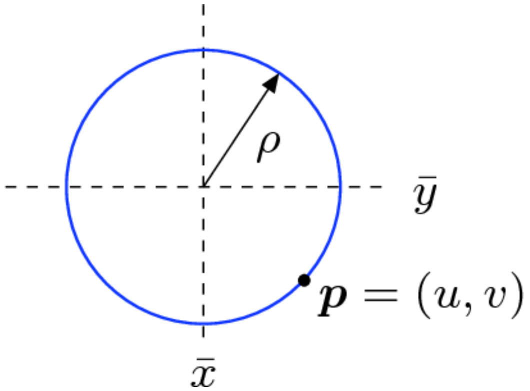
- Hierarhična Houghova transformacija
  - Za veliko natančnost potrebujemo veliko akumulacijsko polje
    - Več kotov upoštevamo, večja je njihova natančnost, ampak tudi računska kompleksnost!
  - Rešitev: hierarhično računanje
    - Najprej pri majhni ločljivosti poiščemo premice v grobem
    - Nato za vsako premico poiščemo natančno lokacijo v večjem ak. Polju z večjo natančnostjo

Houghova transformacija za krožnice

- Krožnico v 2D opišemo s tremi parametri:
  - Koordinati središča ter polmer

$$Circle = \langle \bar{x}, \bar{y}, \rho \rangle$$

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

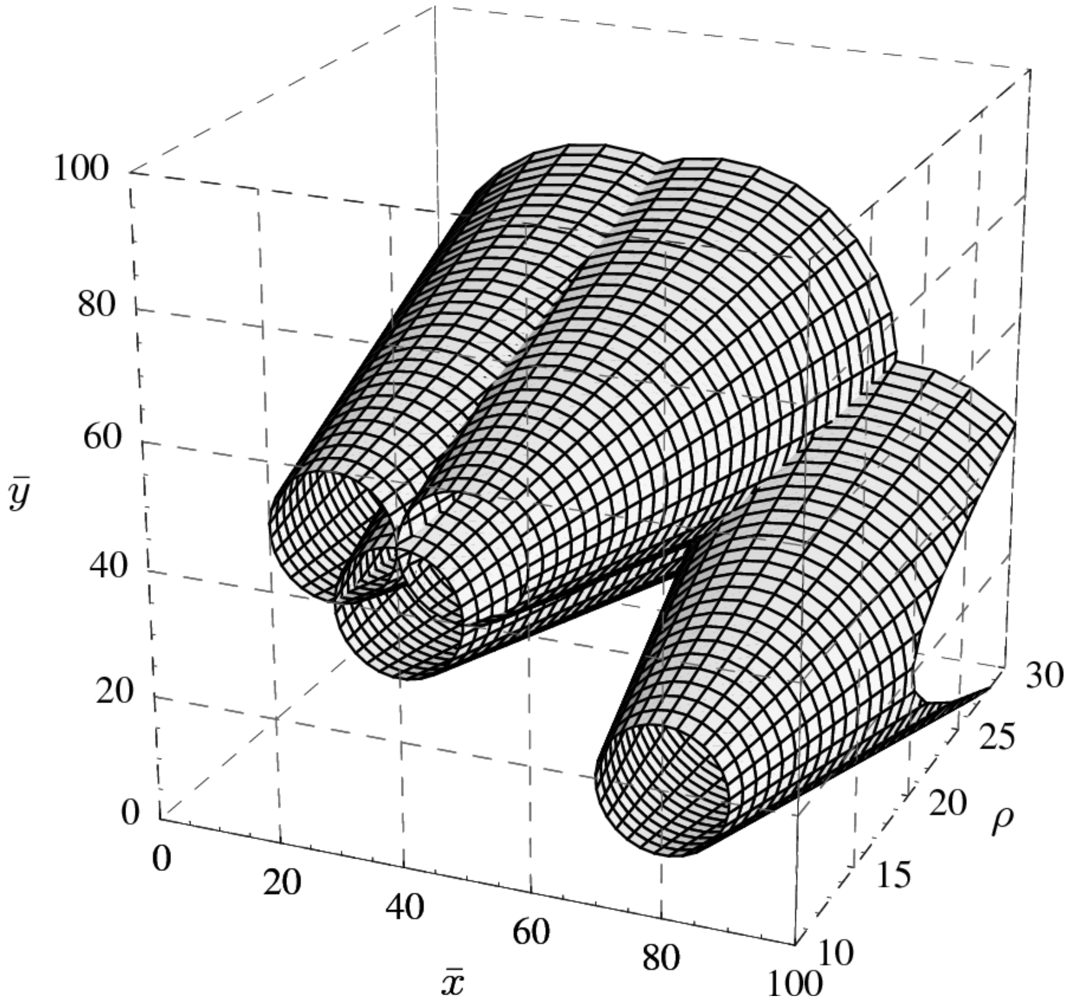


- Iščemo tri parametre

3D akumulatorsko polje

- 3D parametrični prostor

=> 3D akumulatorsko polje



Algoritem za krožnice

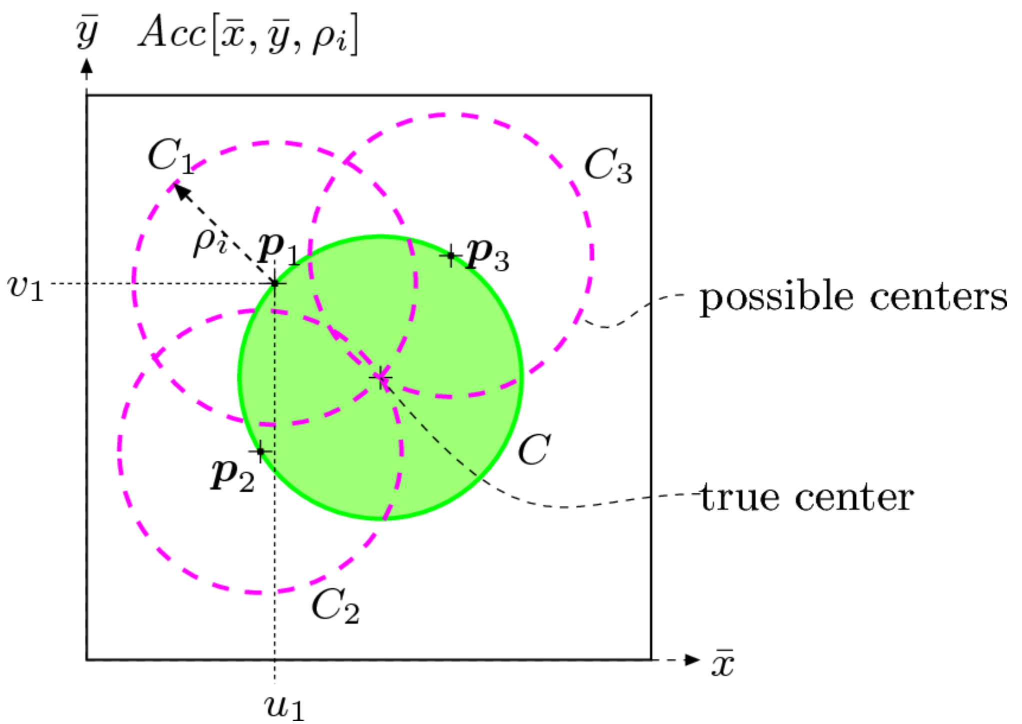
HoughCircles(I)

Returns the list of parameters $\langle \bar{x}_i, \bar{y}_i, \rho_i \rangle$ corresponding to the strongest circles found in the binary image I.

Set up a three-dimensional array $Acc[\bar{x}, \bar{y}, \rho]$ and initialize to 0
for all image coordinates (u, v) **do**
  if $I(u, v)$ is an edge point **then**
    for all $(\bar{x}_i, \bar{y}_i, \rho_i)$ in the accumulator space **do**
      if $(u - \bar{x}_i)^2 + (v - \bar{y}_i)^2 = \rho_i^2$ **then**
        Increment $Acc[\bar{x}_i, \bar{y}_i, \rho_i]$
$MaxCircles \leftarrow$ FindMaxCircles(Acc) ▷ a list of tuples $\langle \bar{x}_j, \bar{y}_j, \rho_j \rangle$
return $MaxCircles$.

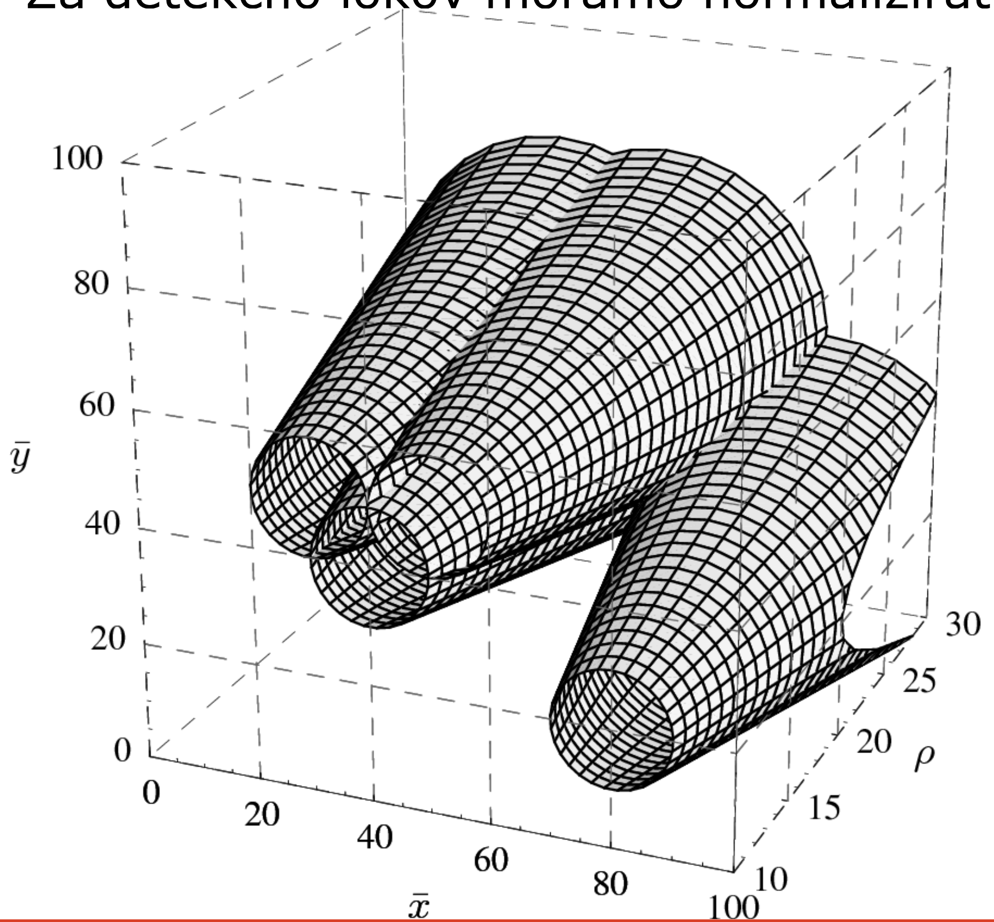
Algoritem za krožnice

- Kako izračunati indeks akumulatorskega polja:
  - Preverimo vse celice 3D akumulatroskega polja
    - 3D iskanje, zelo potratno
  - Za vsako središče izračunamo radij
    - 2D iskanje
  - Za vsak radij izračunamo možna središča
    - „generiramo krožnice" v akumulatorskem polju
    - 1D iskanje, najhitrejše



3D akumulatorsko polje

- Vsaka točka na krožnici na sliki se preslika v stožec v 3D akumulatorskem polju
  - Iščemo presečišča stožcev
  - Za detekcijo lokov moramo normalizirati vrednosti v ak. polju

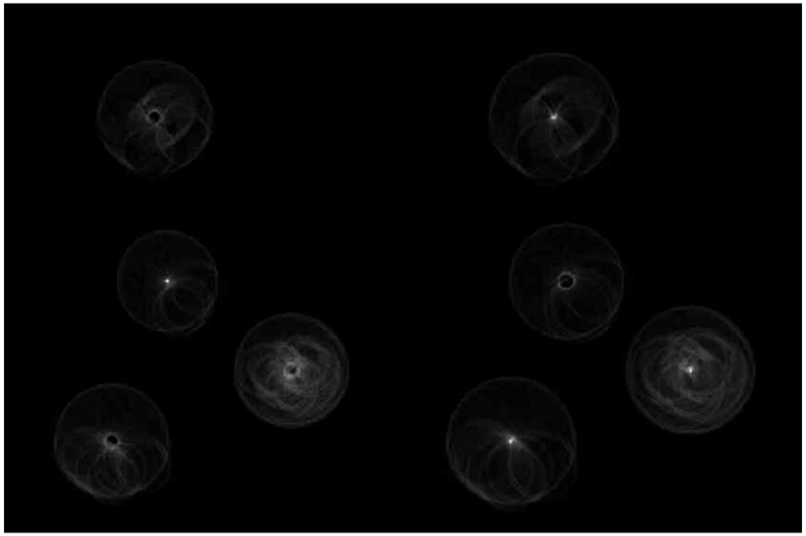


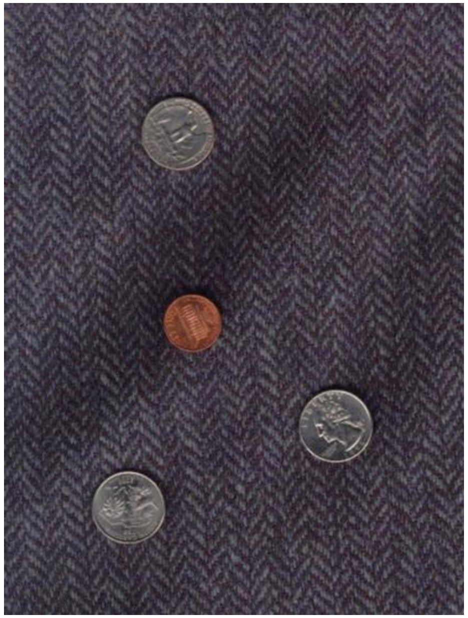
3D parameter space:
$\bar{x}, \bar{y} = 0 \dots 100$
$\rho = 10 \dots 30$

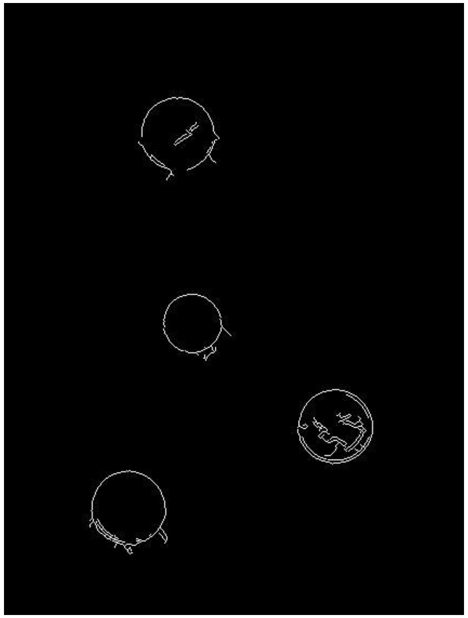
Image points $\boldsymbol{p}_k$:
$\boldsymbol{p}_1 = (30, 50)$
$\boldsymbol{p}_2 = (50, 50)$
$\boldsymbol{p}_3 = (40, 40)$
$\boldsymbol{p}_4 = (80, 20)$

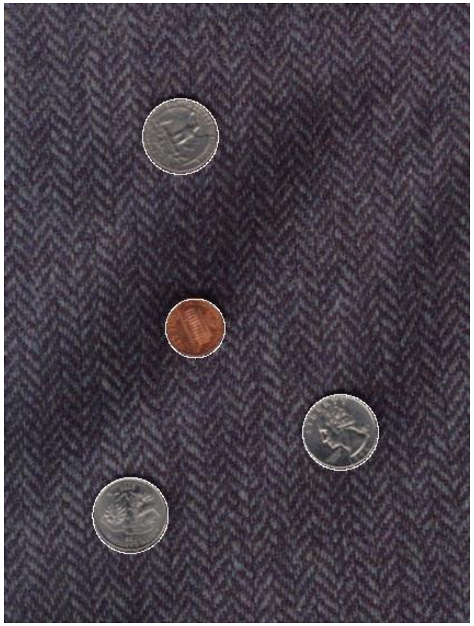
Primer

- Če je radij krožnic poznan, lahko iščemo samo središča
- Primer:





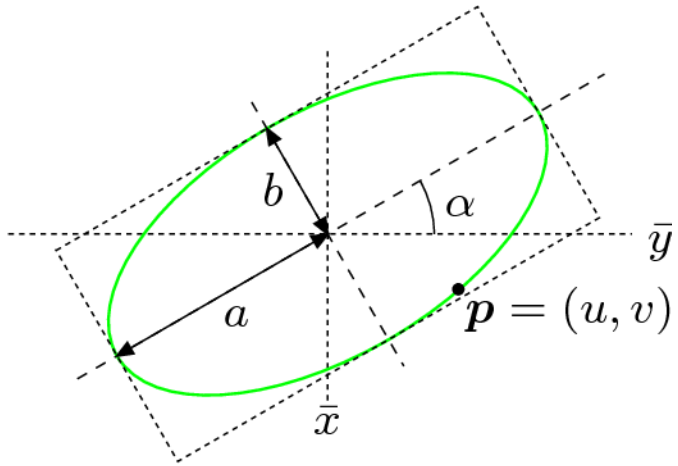




Detekcija elips

- Krožnice v 3D se preslikajo v elipse v 2D
- Elipso v 2D opišemo s petimi parametri:

$$Ellipse = \langle \bar{x}, \bar{y}, r_a, r_b, \alpha \rangle$$

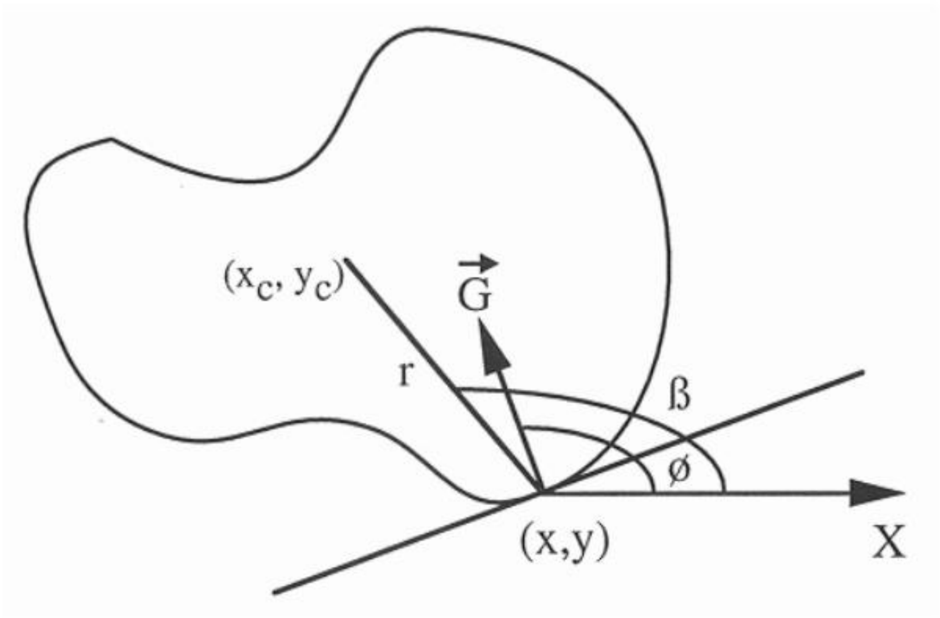


- Potrebujemo 5D akumulatorsko polje!
  - Ogromna prostorska zahtevnost

- Boljša uporaba Posplošene Houghove transformacije

Posplošena Houghova transformacija

- Primerna za katerikoli parametrizirano obliko
- Točke na robu zakodiramo z razdaljo in kotom do referenčne točke ob upoštevanju gradienta v dani točki
- Zgradimo r-tabelo:



$\phi_1 = 0$	$(r,\beta)_{1_1}$	$(r,\beta)_{1_2}$	$\cdots$	$(r,\beta)_{1_{n_1}}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
ϕ_j	$(r,\beta)_{j_1}$	$(r,\beta)_{j_2}$	$\cdots$	$(r,\beta)_{j_{n_1}}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\phi_k = \pi$	$(r,\beta)_{k_1}$	$(r,\beta)_{k_2}$	$\cdots$	$(r,\beta)_{k_{n_1}}$

Posplošena Houghova transformacija

- 4D akumualtorsko polje $H(x_c, y_c, S, \theta)$
  - Ref. točka + skala + orientacija

- Algoritem:
  1. Za vsako točko (x, y) kjer je $|G(x,y)| > T$:
  - Poišči vrstico v r-tabeli, kjer je $\phi_j = \angle G(x,y)$
  - Potem za vsak par $(r, \beta)_i$ ($i = 1, \cdots, n_j$) iz te vrstice in za vse vrednosti S in θ poišči

$$x_c = x + r\ S\ cos(\beta + \theta)$$
$$y_c = y + r\ S\ sin(\beta + \theta)$$

  ter povečaj vrednost ustreznega akumulatorskega polja

$$H(x_d, y_c, S, \theta) = H(x_c, y_c, S, \theta) + 1$$

  2. Na koncu poiščemo maksimalne vrednosti v akumulacijskem polju $H(x_c, y_c, S, \theta)$