

# Robotika in računalniško zaznavanje (RRZ)

## Detekcija enostavnih krivulj

Danijel Skočaj

Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

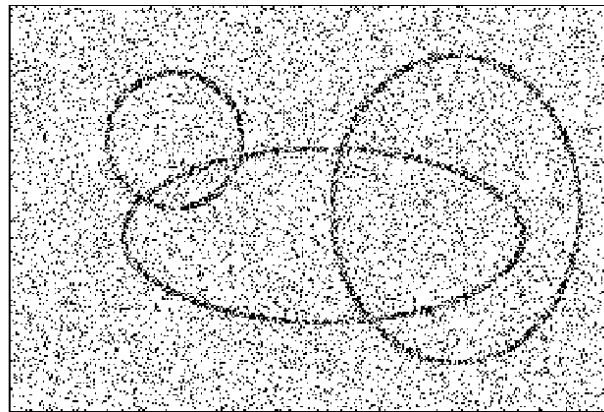
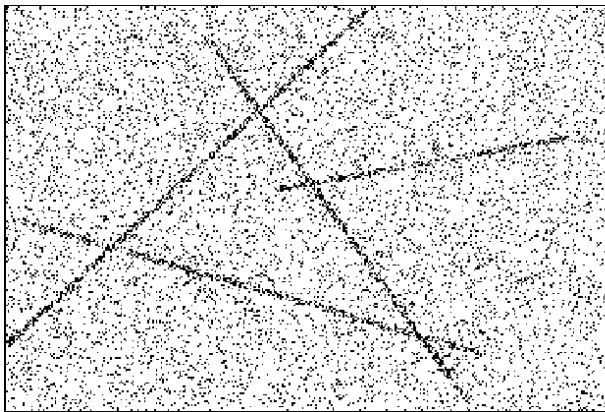
Literatura: W. Burger, M. J. Burge (2008)

Digital Image Processing, poglavje 9

v7.0

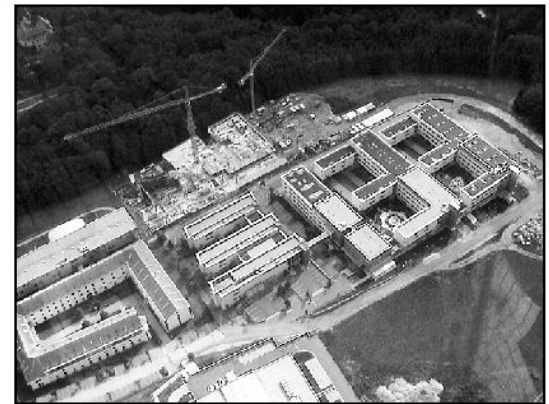
# Enostavne krivulje

- 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

---

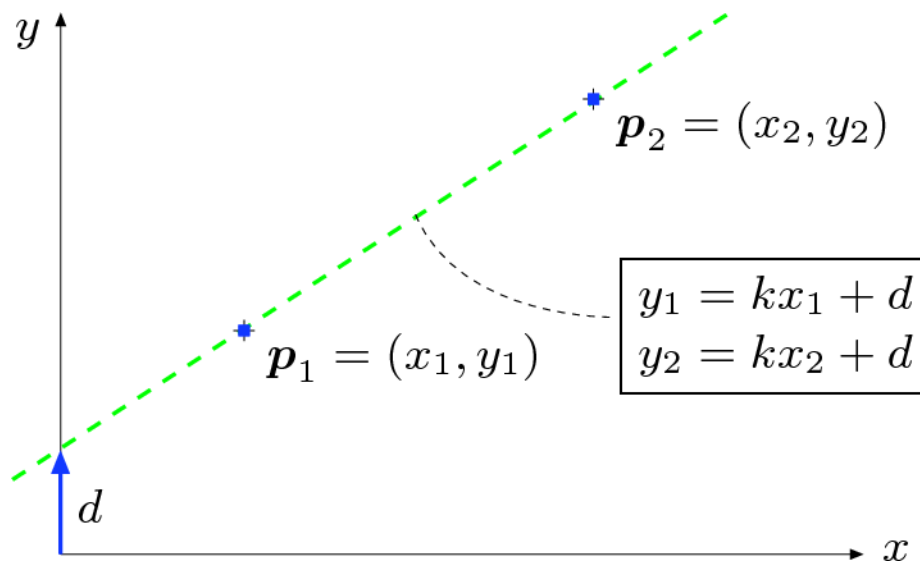
- Globalna metoda za detekcijo premic, krožnic, drugih parametričnih krivulj
- Najpogosteje uporabljana za detekcijo premic
- Na sliki je premica, če na njej leži veliko število robnih slikovnih elementov
- Naivna ideja: zgeneriraj vse možne premice in preveri koliko robnih slikovnih elementov leži na 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  $p_1 = (x_1, y_1)$  and  $p_2 = (x_2, y_2)$  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



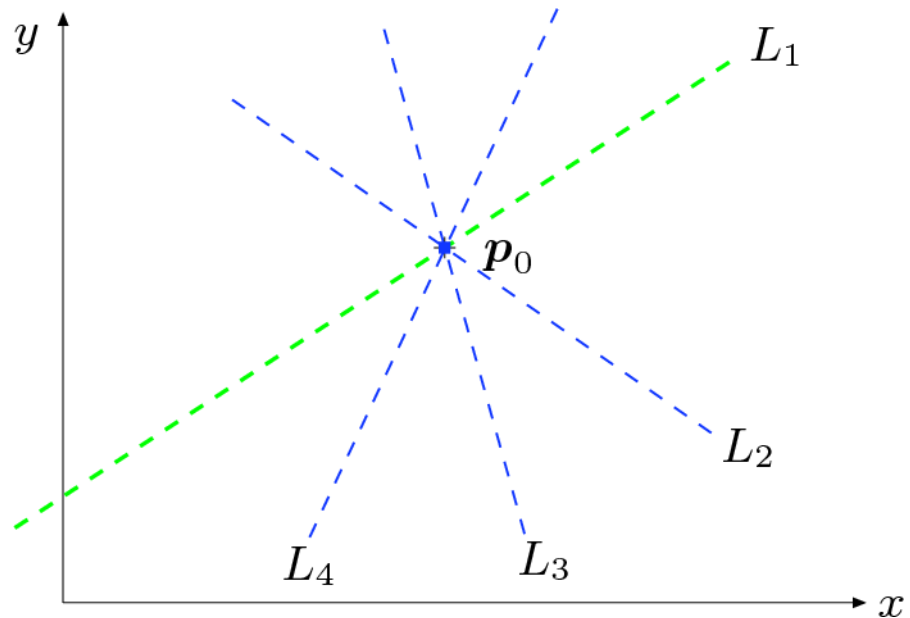
# Parametrični oz. Houghov prostor

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

- Skozi točko  $p_0$  :

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

$$d_j = -x_0 k_j + y_0$$

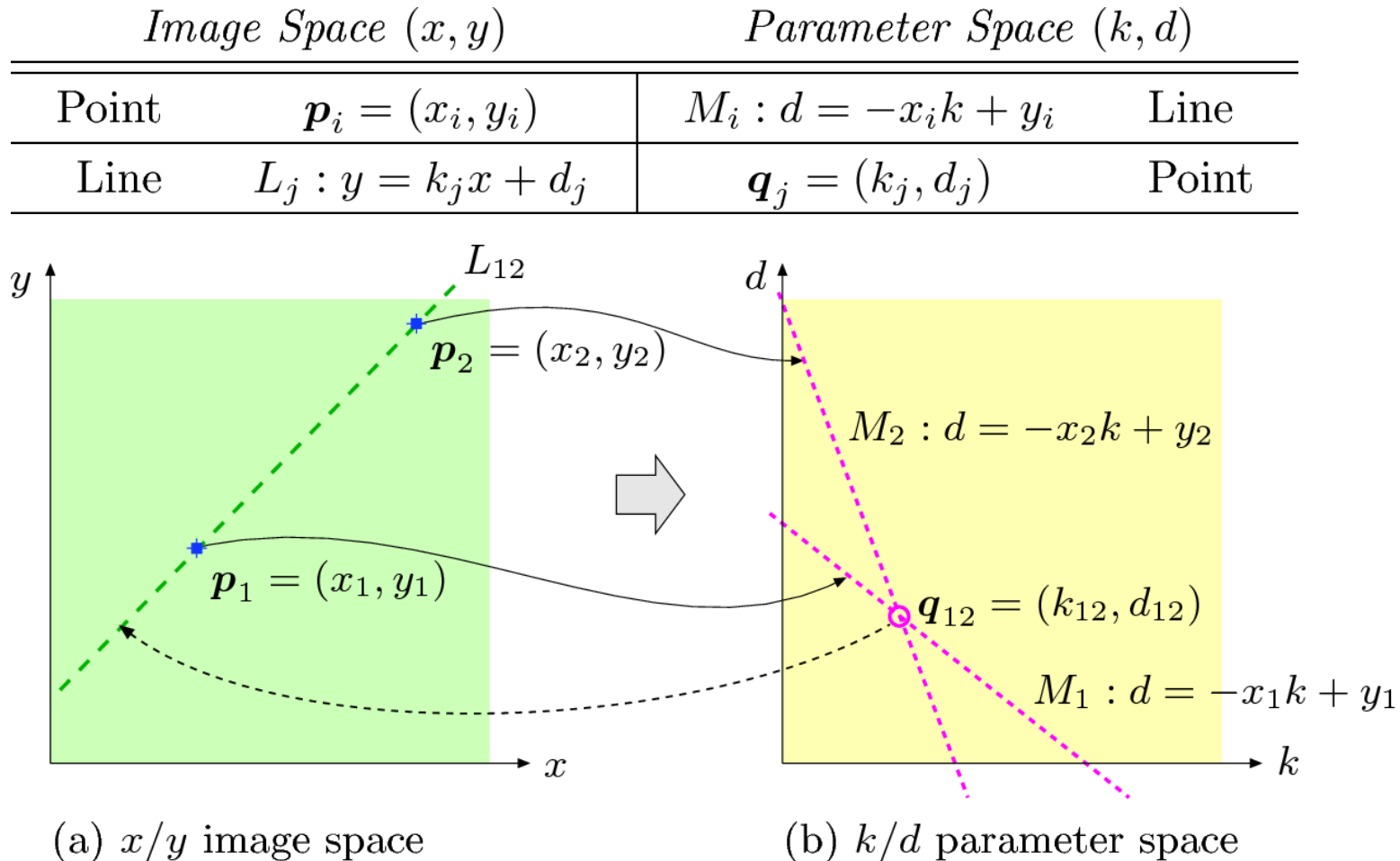


- Skozi poljubno točko  $p_i$  :

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

- $x_i$  in  $y_i$  sta parametra parametričnem prostoru, ki ga napenjata  $k$  in  $d$ !

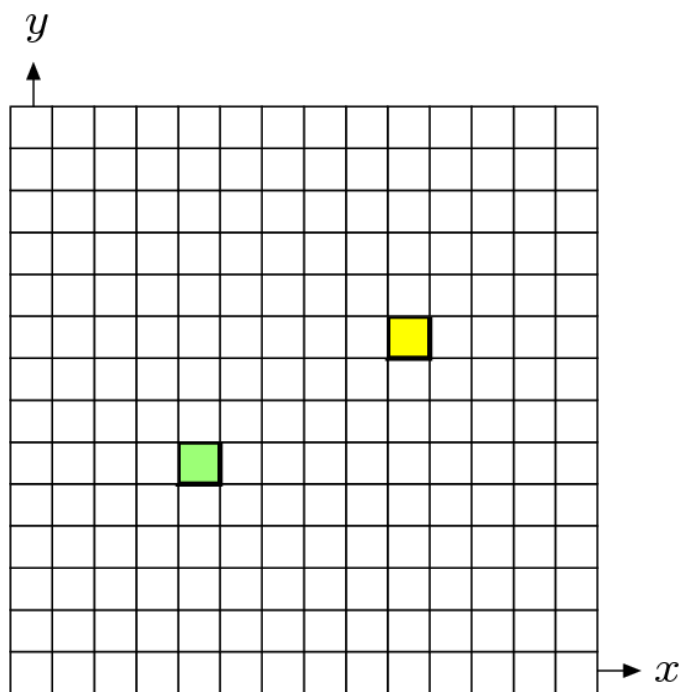
# Prostora slik in parametrov



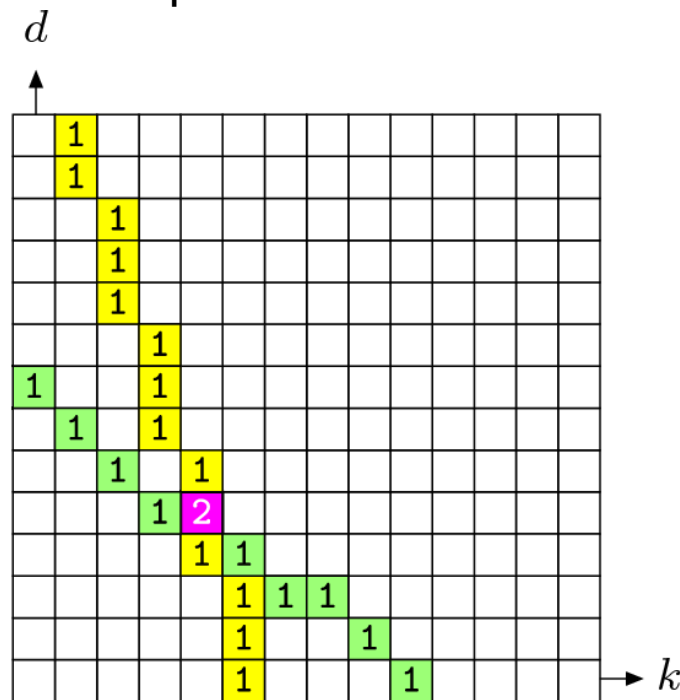
- Presečišče  $N$  premic v prostoru parametrov na lokaciji  $(k', d')$   
 -> v prostoru slik  $N$  točk leži na premici  $y = k'x + d'$

# Akumulatorsko polje

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



(a) Image Space

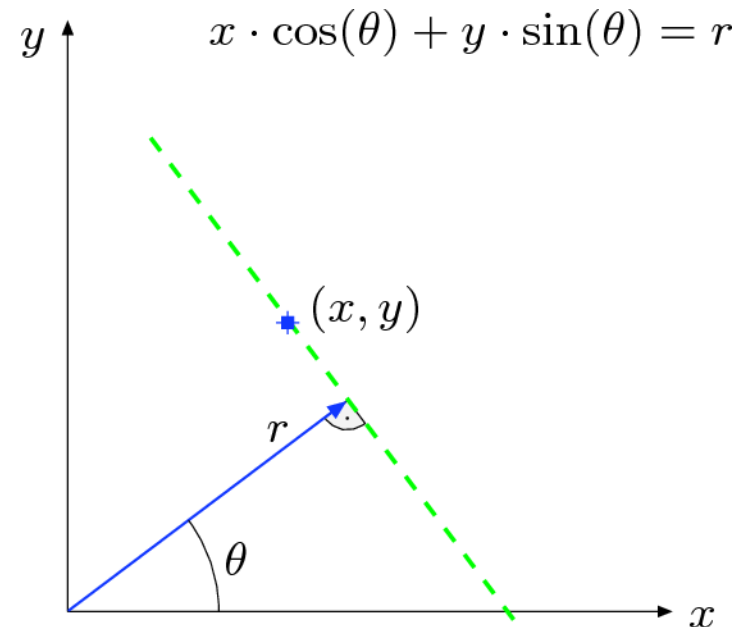
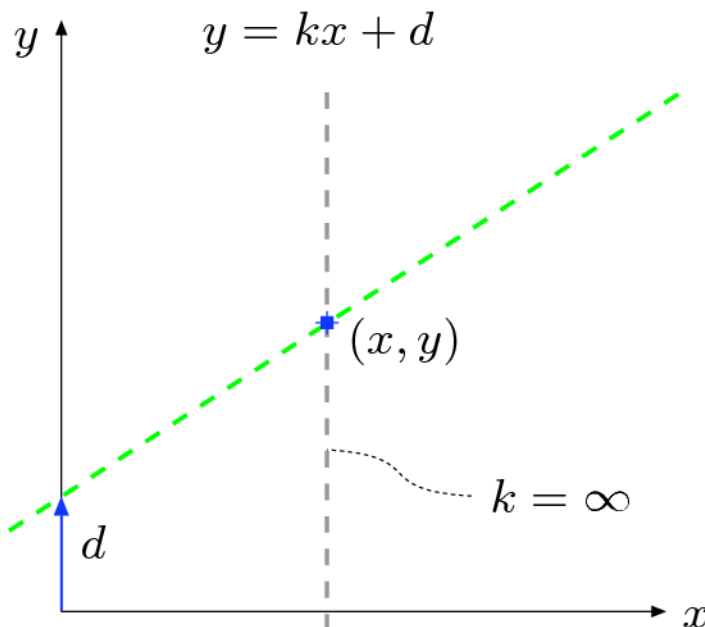


(b) Accumulator Array



# 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  $\theta$  (radij in kot)
  - Linearno spreminjanje obeh parametrov
  - Ni singularnosti



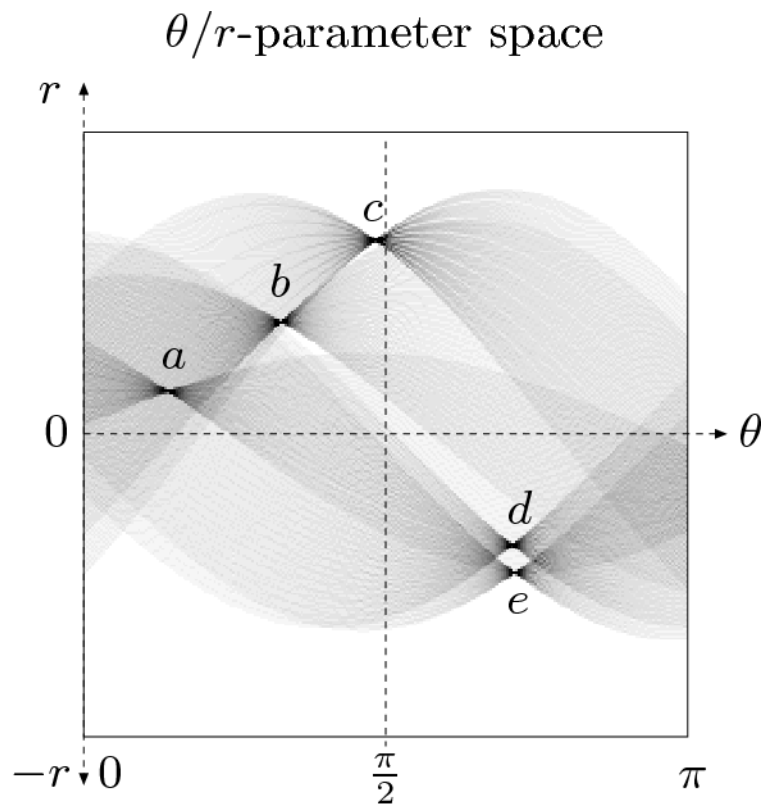
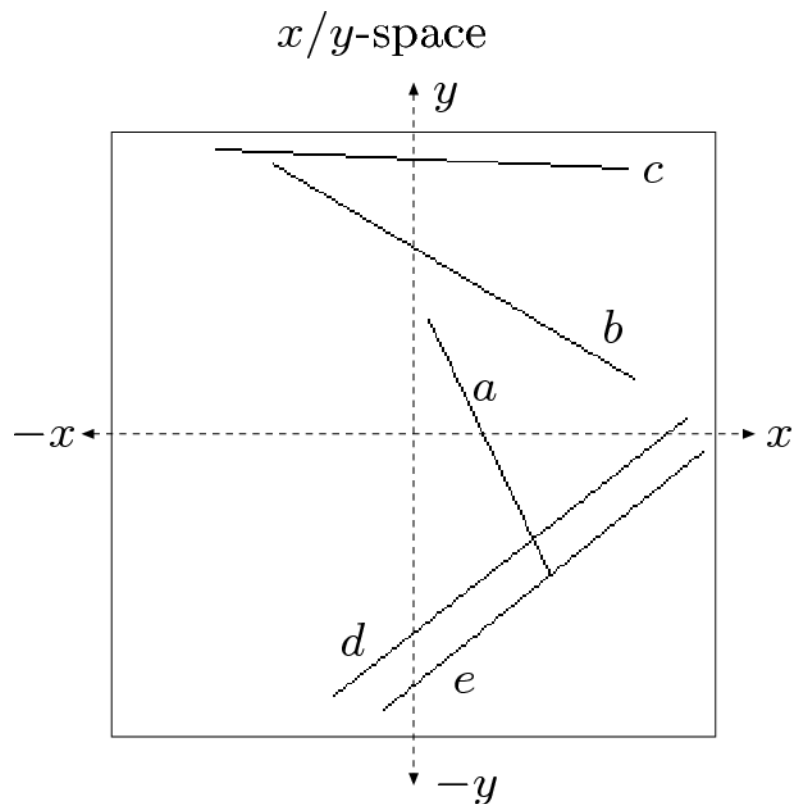
# Akumulatorsko polje

- Akumulatorsko polje v HNF

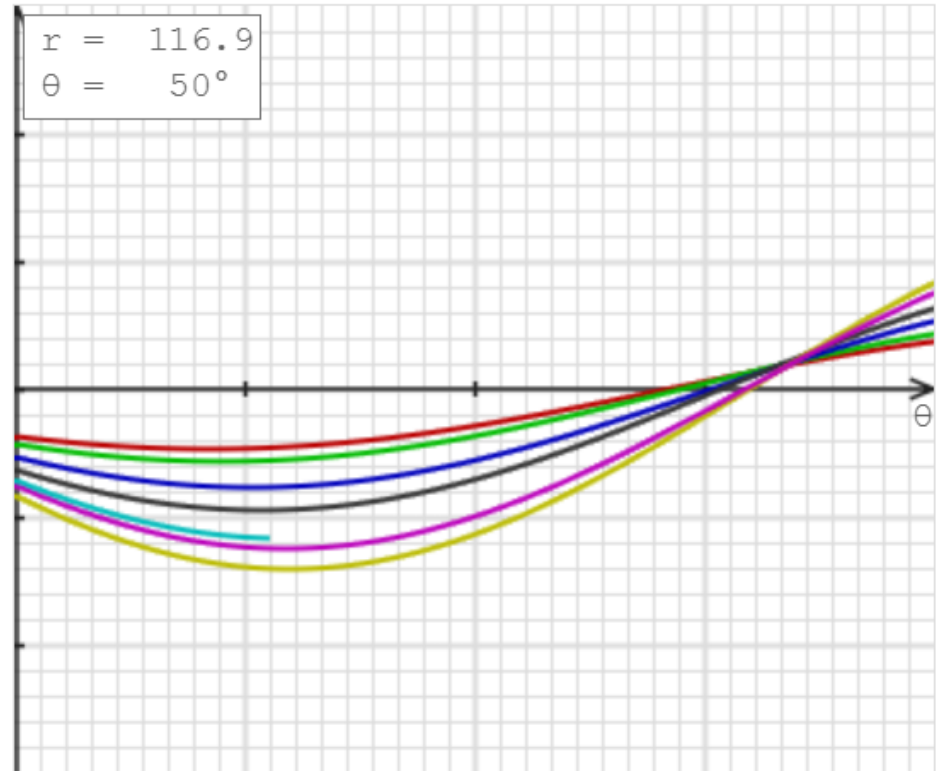
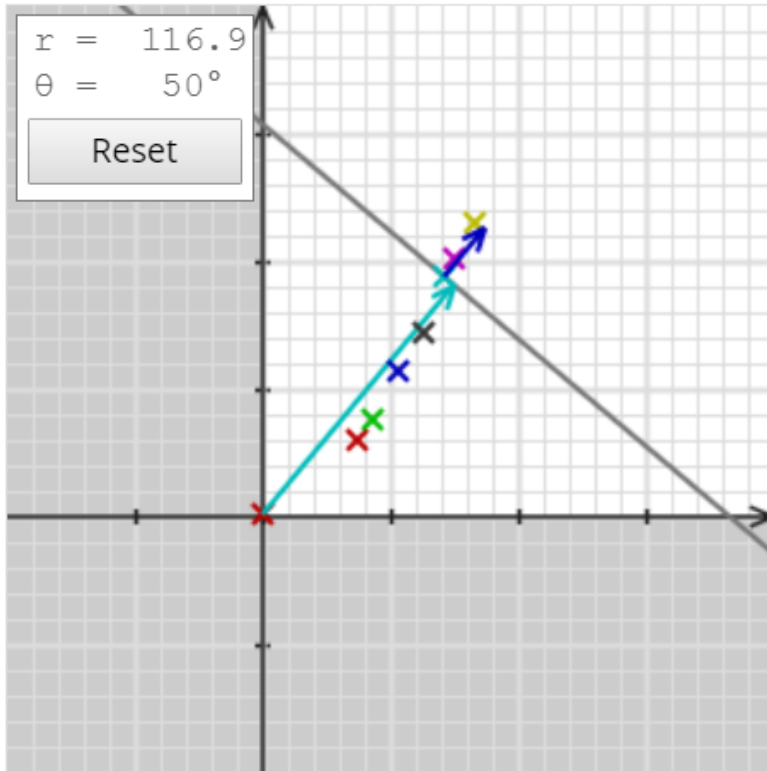
- Računamo radij:  $r_{x_i, y_i}(\theta) = x_i \cdot \cos(\theta) + y_i \cdot \sin(\theta)$

- Meje: kot:  $0 \leq \theta < \Pi$

radij:  $-r_{\max} \leq r_{x,y}(\theta) \leq r_{\max}$ , where  $r_{\max} = \frac{1}{2}\sqrt{M^2 + N^2}$



# Demo



<http://matlabtricks.com/post-39/understanding-the-hough-transform>

# Algoritem

```
1: HOUGHLINES( $I$ )
   Returns the list of parameters  $\langle \theta_i, r_i \rangle$  corresponding to the strongest
   lines found in the binary image  $I$ .

2:   Set up a two-dimensional array  $Acc[\theta, r]$  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
       Get coordinate relative to the image center  $(u_c, v_c)$ :
6:        $(x, y) \leftarrow (u - u_c, v - v_c)$ 
7:       for  $\theta_i = 0 \dots \pi$  do
8:          $r_i = x \cdot \cos(\theta_i) + y \cdot \sin(\theta_i)$ 
9:         Increment  $Acc[\theta_i, r_i]$ 
   Return the list of parameter pairs  $\langle \theta_j, r_j \rangle$  for  $K$  strongest lines:
10:   $MaxLines \leftarrow \text{FINDMAXLINES}(Acc, K)$ 
11:  return  $MaxLines$ .
```

# Algoritem

- Potrebno je pravilno diskretizirati akumulatorsko polje

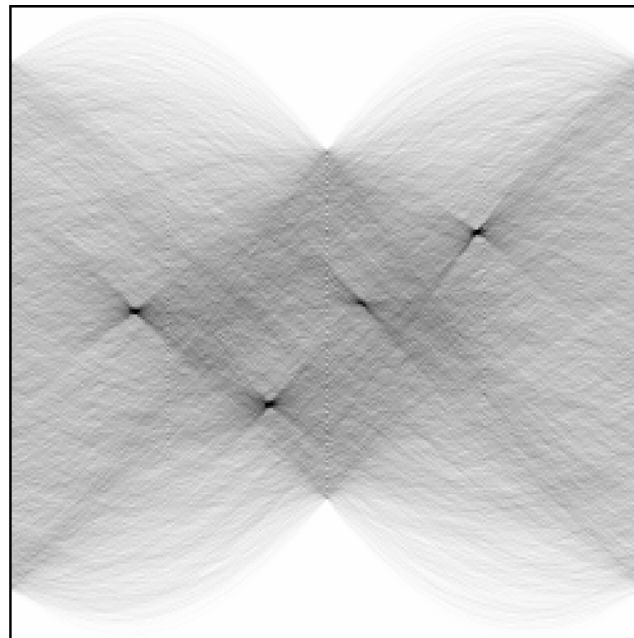
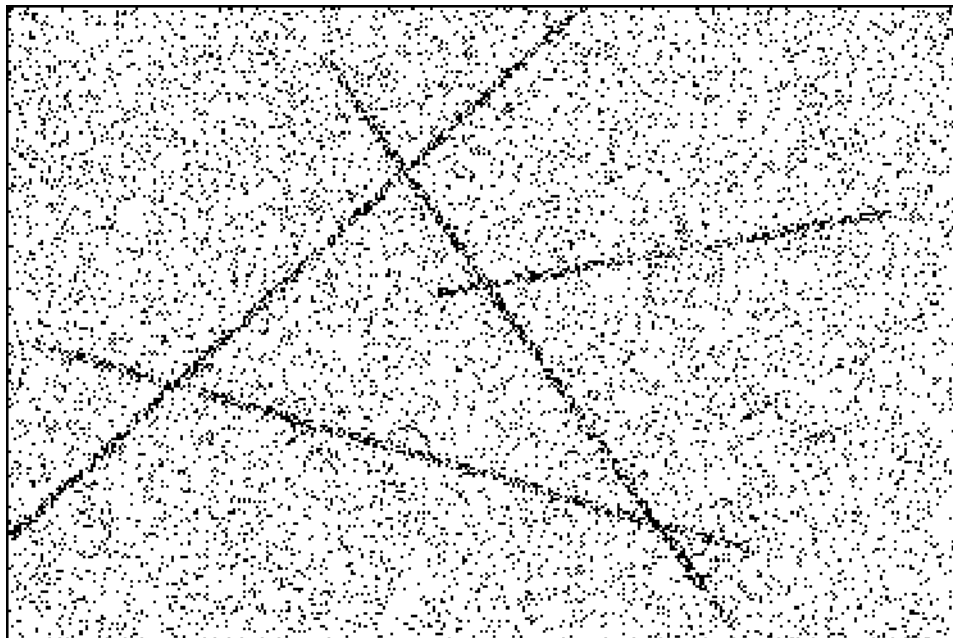
$$\Delta_{\theta} = \frac{\pi}{N_{\theta}} \quad \text{and} \quad \Delta_r = \frac{2 \cdot r_{\max}}{N_r}$$

```
1 class LinearHT {
2   ImageProcessor ip; // reference to the original image I
3   int xCtr, yCtr;    // x/y-coordinates of image center (uc, vc)
4   int nAng;          // Nθ steps for the angle (θ = 0...π)
5   int nRad;          // Nr steps for the radius (r = -rmax...rmax)
6   int cRad;          // center of radius axis (r = 0)
7   double dAng;        // increment of angle Δθ
8   double dRad;        // increment of radius Δr
9   int[][] houghArray; // Hough accumulator Acc[θi, ri]
10
11  //constructor method:
12  LinearHT(ImageProcessor ip, int nAng, int nRad) {
13    this.ip = ip;
14    this.xCtr = ip.getWidth()/2;
15    this.yCtr = ip.getHeight()/2;
16    this.nAng = nAng;
17    this.dAng = Math.PI / nAng;
18    this.nRad = nRad;
19    this.cRad = nRad / 2;
20    double rMax = Math.sqrt(xCtr * xCtr + yCtr * yCtr);
21    this.dRad = (2.0 * rMax) / nRad;
22    this.houghArray = new int[nAng][nRad];
23    fillHoughAccumulator();
24  }
25}
```

```
26 void fillHoughAccumulator() {
27   int h = ip.getHeight();
28   int w = ip.getWidth();
29   for (int v = 0; v < h; v++) {
30     for (int u = 0; u < w; u++) {
31       if (ip.get(u, v) > 0) {
32         doPixel(u, v);
33       }
34     }
35   }
36 }
37
38 void doPixel(int u, int v) {
39   int x = u - xCtr, y = v - yCtr;
40   for (int i = 0; i < nAng; i++) {
41     double theta = dAng * i;
42     int r = cRad + (int) Math rint
43       ((x*Math.cos(theta) + y*Math.sin(theta)) / dRad);
44     if (r >= 0 && r < nRad) {
45       houghArray[i][r]++;
46     }
47   }
48 }
49
50 } // end of class LinearHT
```

# Analiza akumulatorskega polja

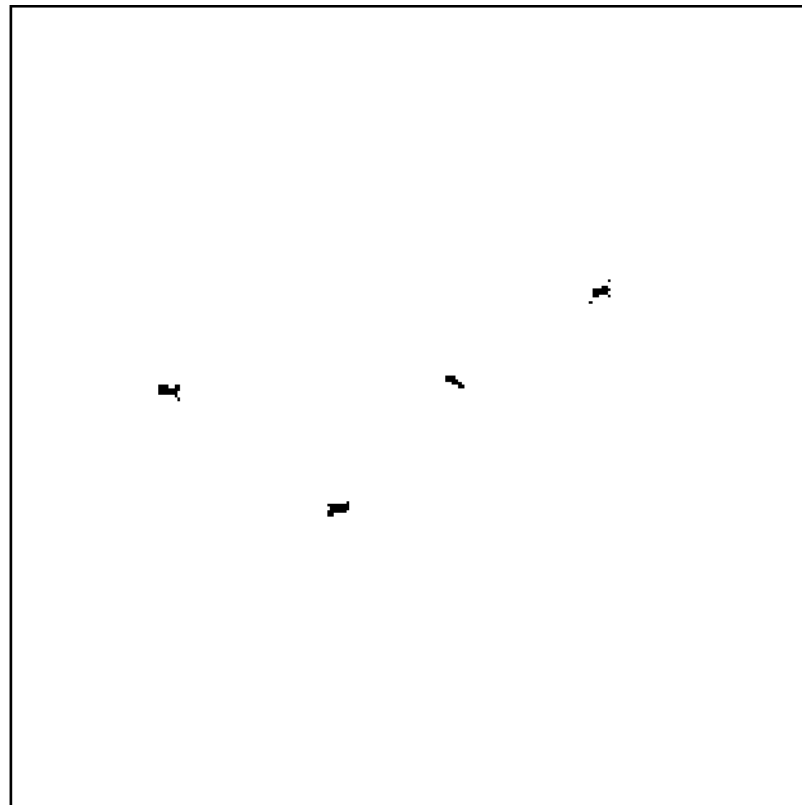
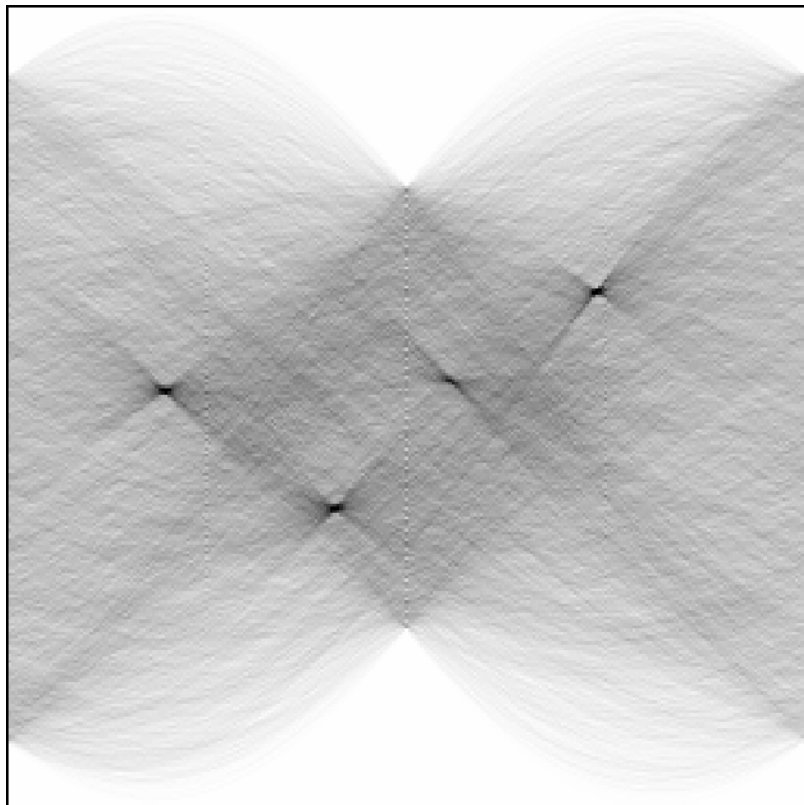
- Detekcija maksimumov v akumulatorskem polju
  - So „razmazani“, lokalni maksimumi
  - Zaradi šuma, diskretizacije



- Dva pristopa:
  - Upragovljenje
  - Dušitev ne-maksimumov

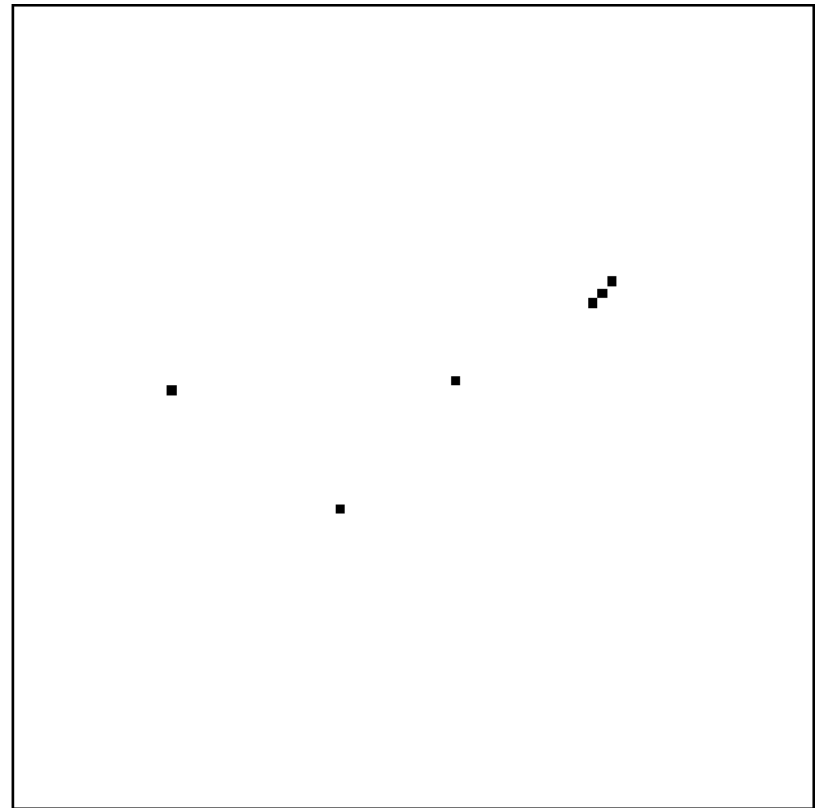
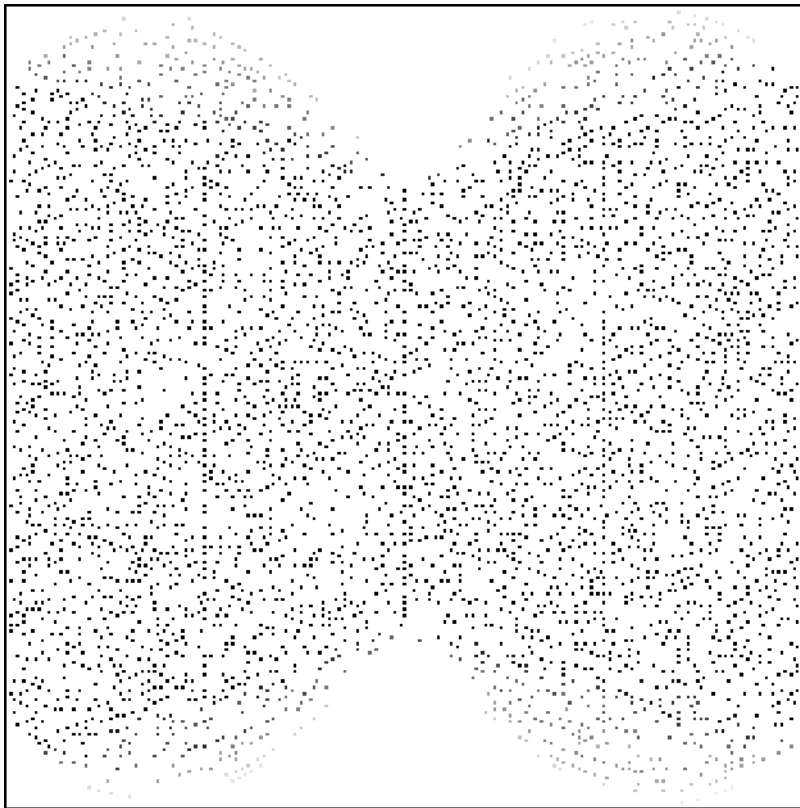
# Upragovljenje

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



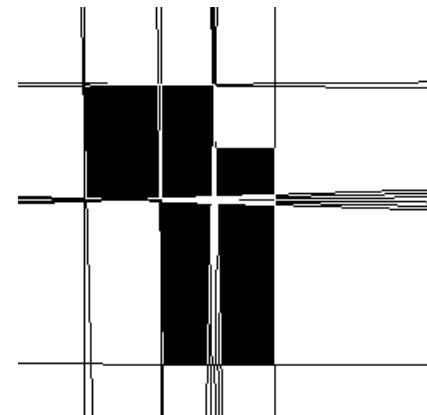
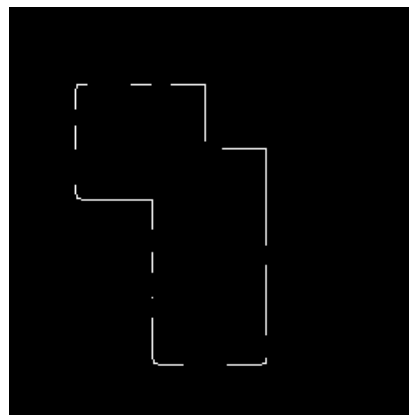
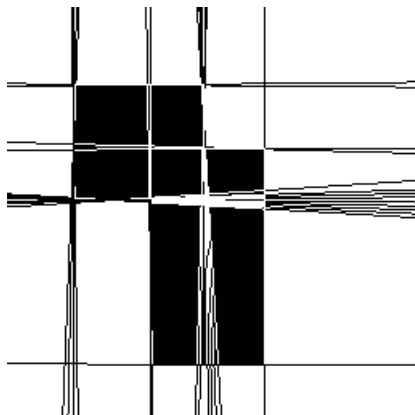
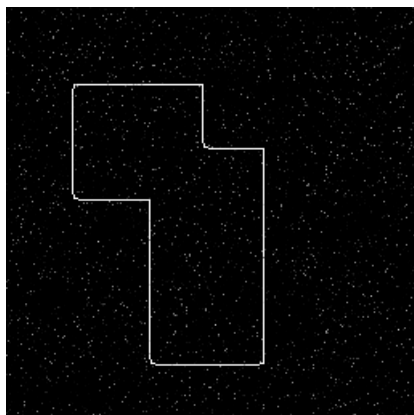
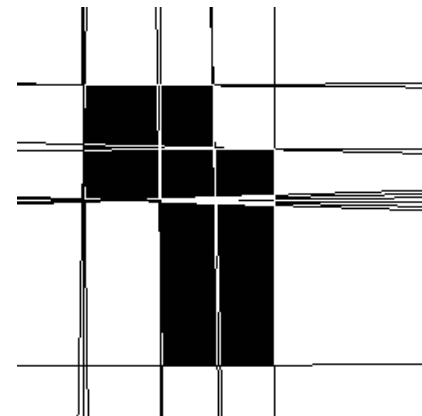
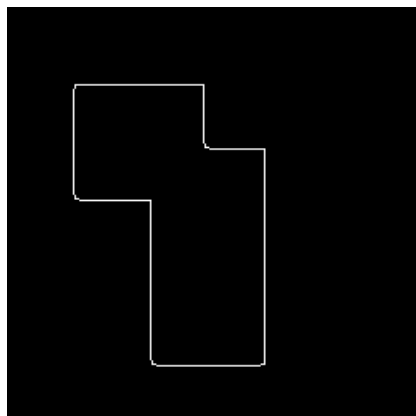
# Dušitev ne-maksimumov

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





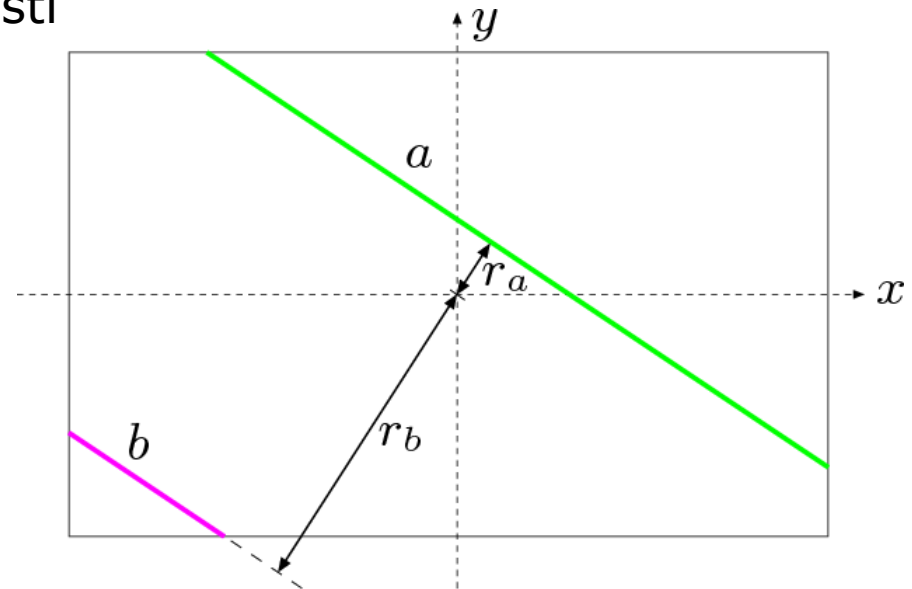
# Primer



# Razširitve

- Izboljšano osveževanje akumulatorskega 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 akumulatorskem polju:

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



- 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:

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

- Iskanje presečišč premic

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

premic  $L_1 = \langle \theta_1, r_1 \rangle$       and       $L_2 = \langle \theta_2, r_2 \rangle$

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:

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

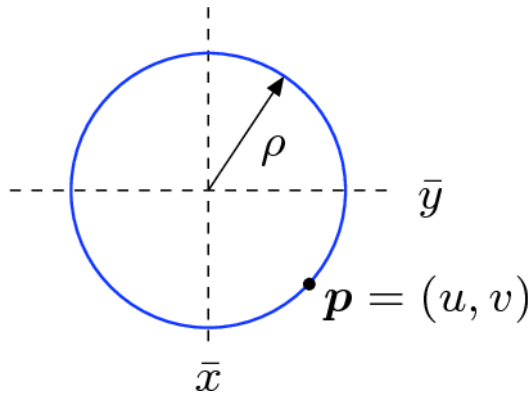
- Upoštevanje moči in smeri roba
  - Uporabljamo sivinsko sliko roba in ne binarno sliko
  - Vrednost akumulatorskega polja povečujemo za vrednosti jakosti roba v danem slikovnem elementu:
$$Acc[\theta, r] \leftarrow Acc[\theta, r] + E(u, v)$$
  - Orientacija roba nam lahko bistveno omeji število kotov za katere računamo radij za posamezno premico
    - Zelo pohitrilo 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

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

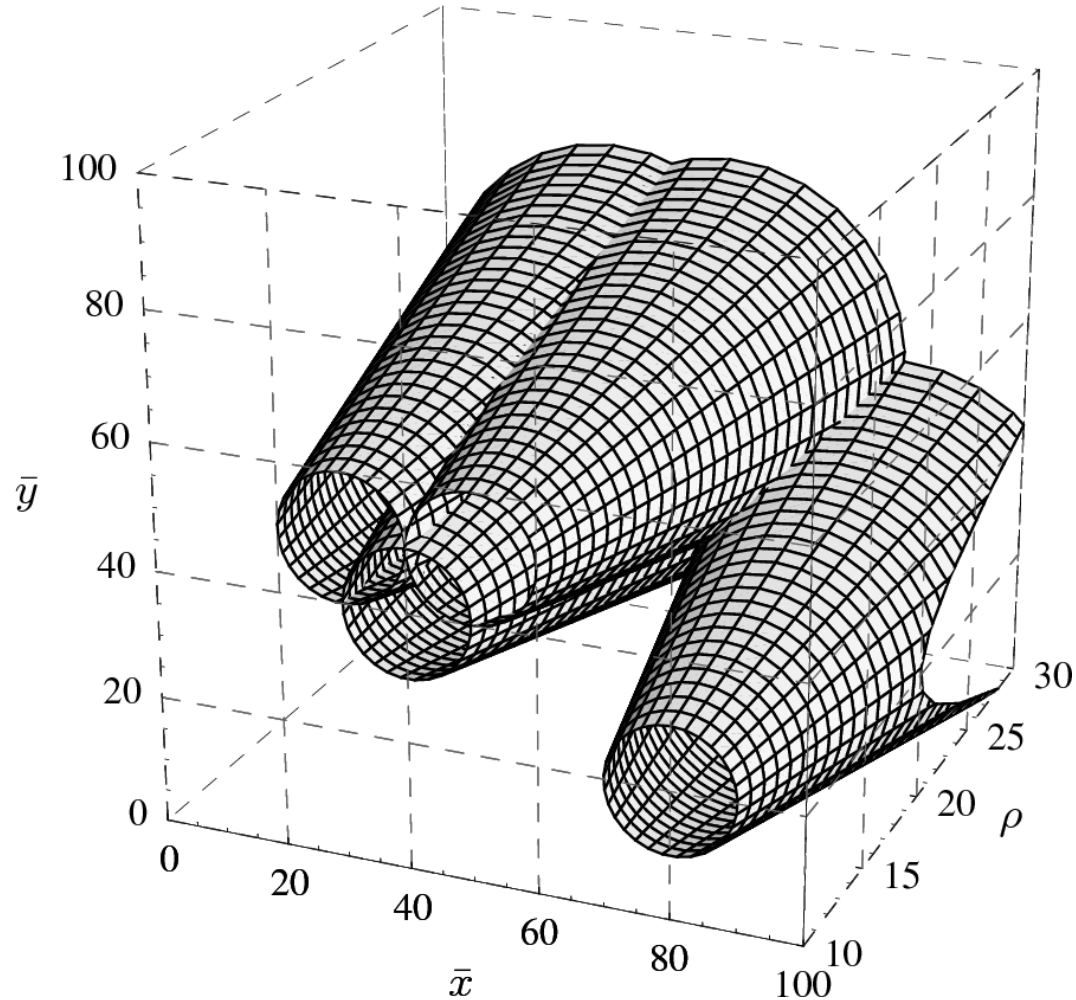
$$(u - \bar{x})^2 + (v - \bar{y})^2 = \rho^2$$



- Iščemo tri parametre

# 3D akumulatorsko polje

- 3D parametrični prostor  
=> 3D akumulatorsko polje



# Algoritem za krožnice

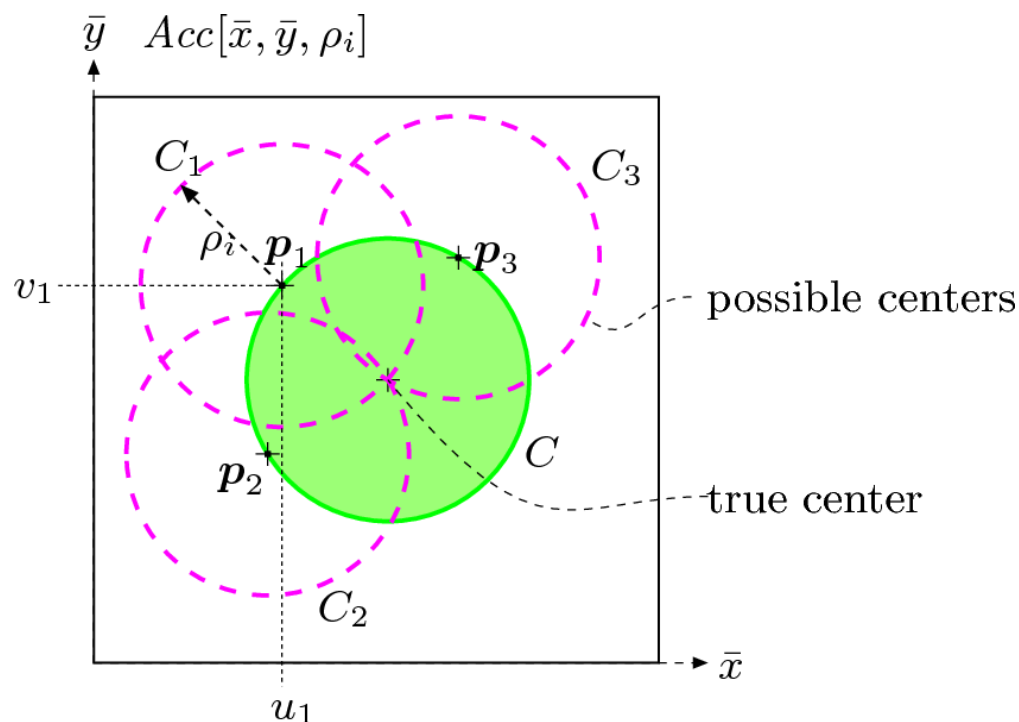
```
1: HOUGH_CIRCLES( $I$ )
   Returns the list of parameters  $\langle \bar{x}_i, \bar{y}_i, \rho_i \rangle$  corresponding to the
   strongest circles found in the binary image  $I$ .

2:   Set up a three-dimensional array  $Acc[\bar{x}, \bar{y}, \rho]$  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  $(\bar{x}_i, \bar{y}_i, \rho_i)$  in the accumulator space do
6:         if  $(u - \bar{x}_i)^2 + (v - \bar{y}_i)^2 = \rho_i^2$  then
7:           Increment  $Acc[\bar{x}_i, \bar{y}_i, \rho_i]$ 
8:    $MaxCircles \leftarrow \text{FIND\_MAX\_CIRCLES}(Acc)$   $\triangleright$  a list of tuples  $\langle \bar{x}_j, \bar{y}_j, \rho_j \rangle$ 
9:   return  $MaxCircles$ .
```



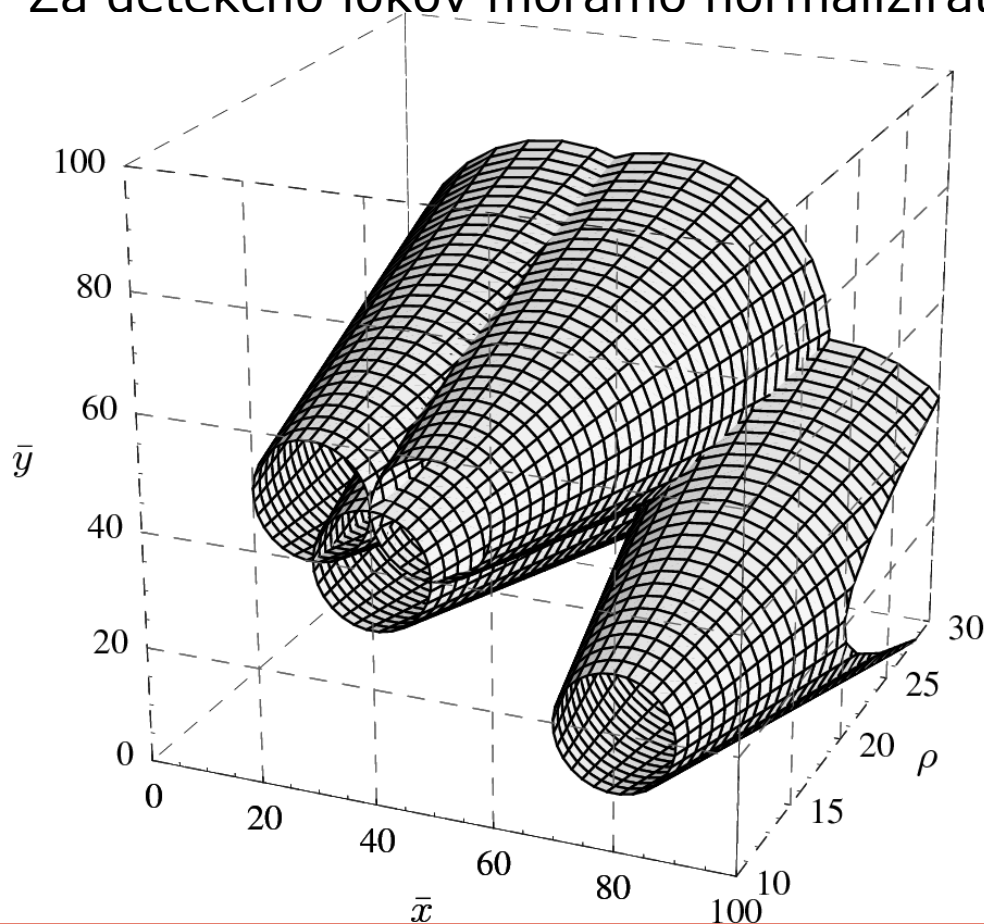
# Algoritem za krožnice

- Kako izračunati indeks akumulatorskega polja:
  - Preverimo vse celice 3D akumulatorskega 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  $\mathbf{p}_k$ :

$$\mathbf{p}_1 = (30, 50)$$

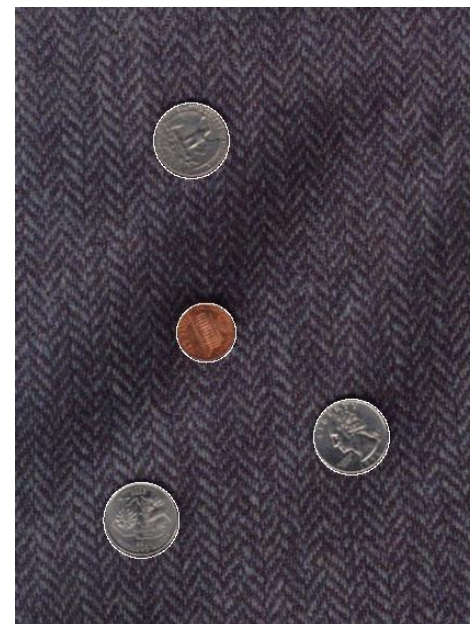
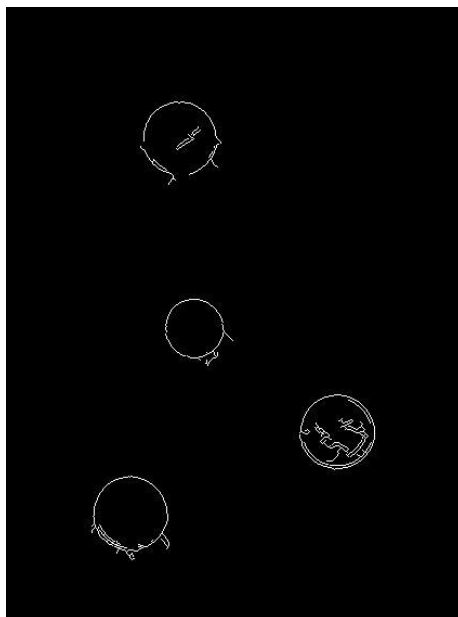
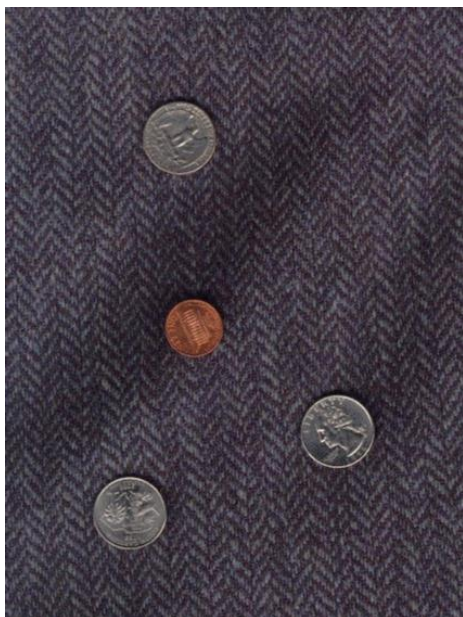
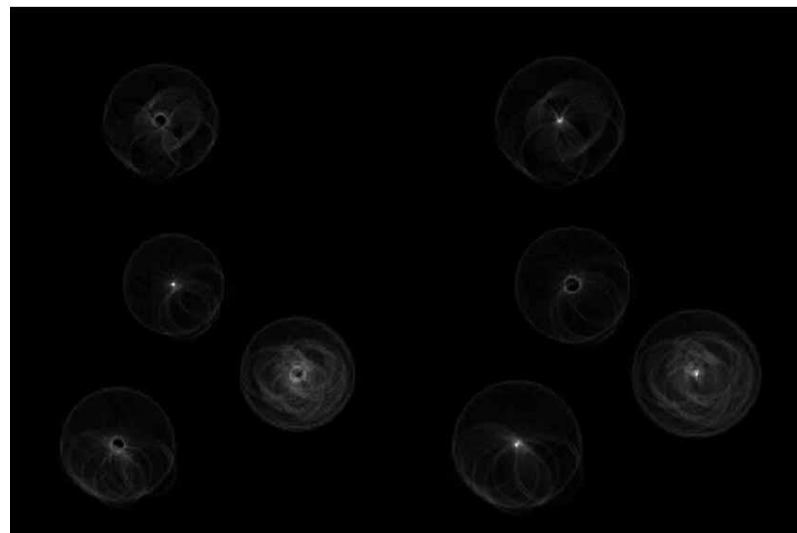
$$\mathbf{p}_2 = (50, 50)$$

$$\mathbf{p}_3 = (40, 40)$$

$$\mathbf{p}_4 = (80, 20)$$

# Primer

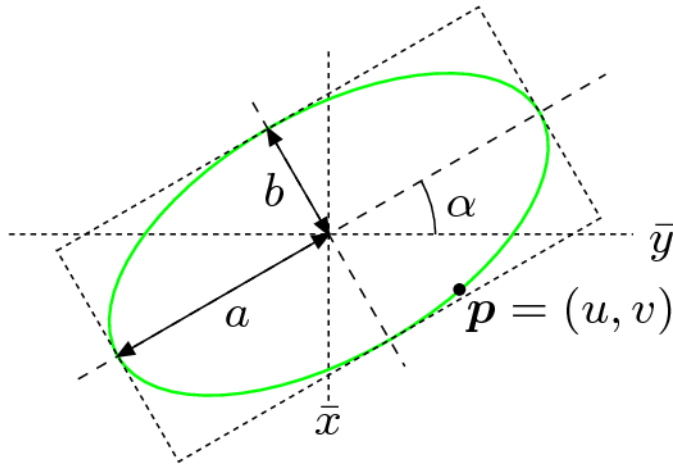
- Če je radij krožnic poznan, lahko iščemo samo središča
- Primer:



# Detekcija elips

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

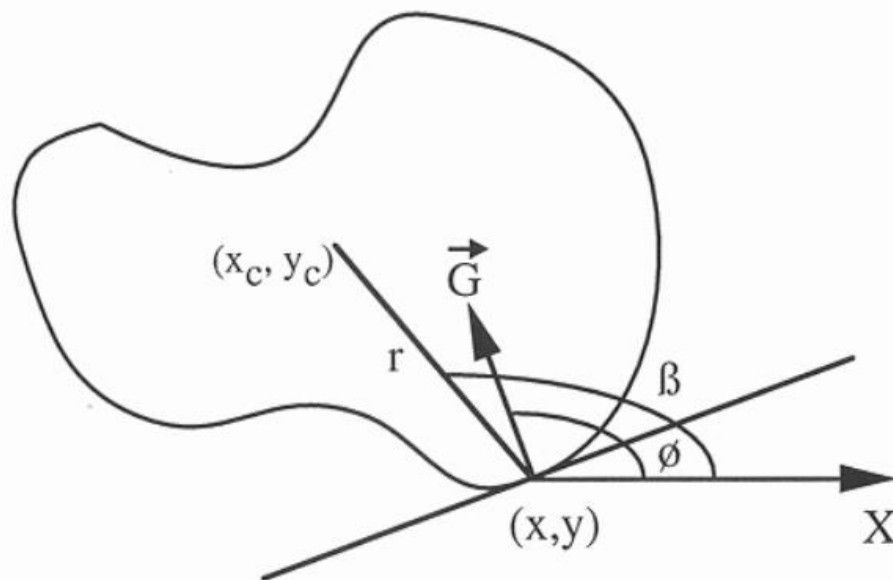
$$Ellipse = \langle \bar{x}, \bar{y}, r_a, r_b, \alpha \rangle$$



- 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:



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

# Posplošena Houghova transformacija

- 4D akumulatorsko 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  $(r, \beta)_i$  ( $i = 1, \dots, n_j$ ) iz te vrstice in za vse vrednosti  $S$  in  $\theta$  poišči

$$\begin{aligned}x_c &= x + r S \cos(\beta + \theta) \\y_c &= y + r S \sin(\beta + \theta)\end{aligned}$$

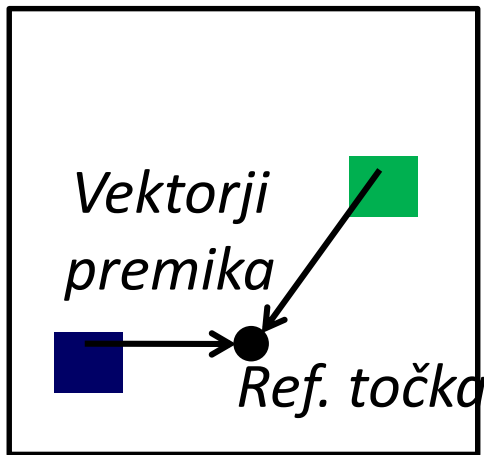
ter povečaj vrednost ustreznega akumulatorskega polja

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

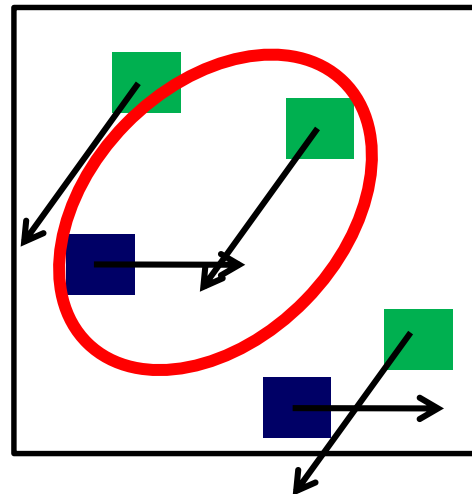
2. Na koncu poiščemo maksimalne vrednosti v akumulacijskem polju  $H(x_c, y_c, S, \theta)$

# Posplošena Houghova transformacija

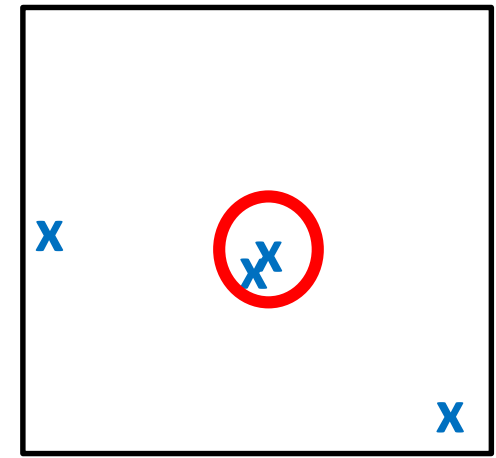
- Intuicija:
  - (barve kodirajo smeri gradienta)



Slika modela



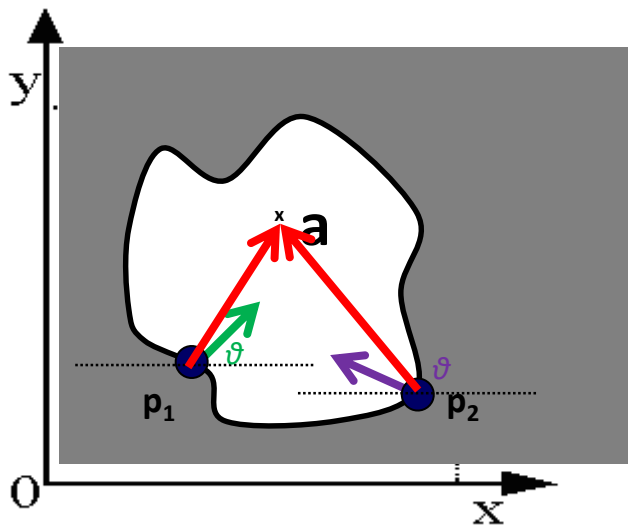
Nova slika



Prostor glasov

# Posplošena Houghova transformacija

- Definiraj model oblike z njenimi robnimi točkami in referenčno točko.



	...
⋮	...

Učenje modela:

V vsaki robni točki izračunaj vektor odmika do referenčne točke:

$$\mathbf{r} = \mathbf{a} - \mathbf{p}_i.$$

Te vrednosti spravi v tabelo, ki jo indeksiramo s smerjo gradienta  $\theta$ .







# Posplošena Houghova transformacija

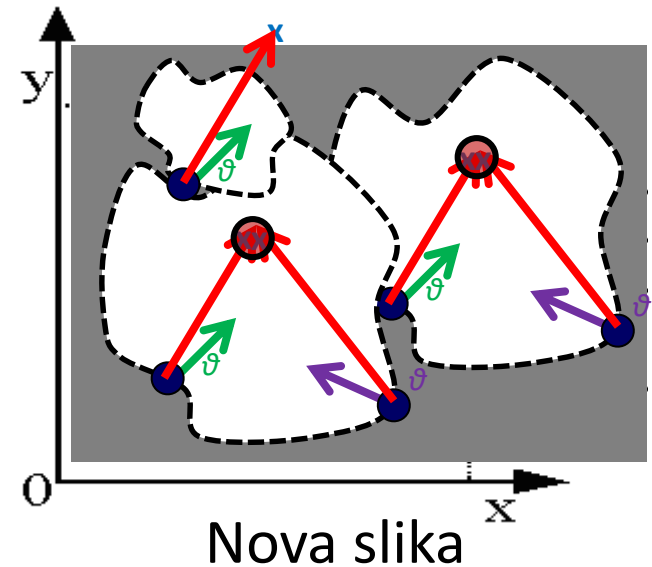
## ▪ Detekcija:

Za vsako robno točko:

- Uporabi njeno orientacijo gradienta  $\theta$  za indeksiranje v shranjeno tabelo.
- Uporabi dobljene vektorje  $r$  za glasovanje položaja referenčne točke.

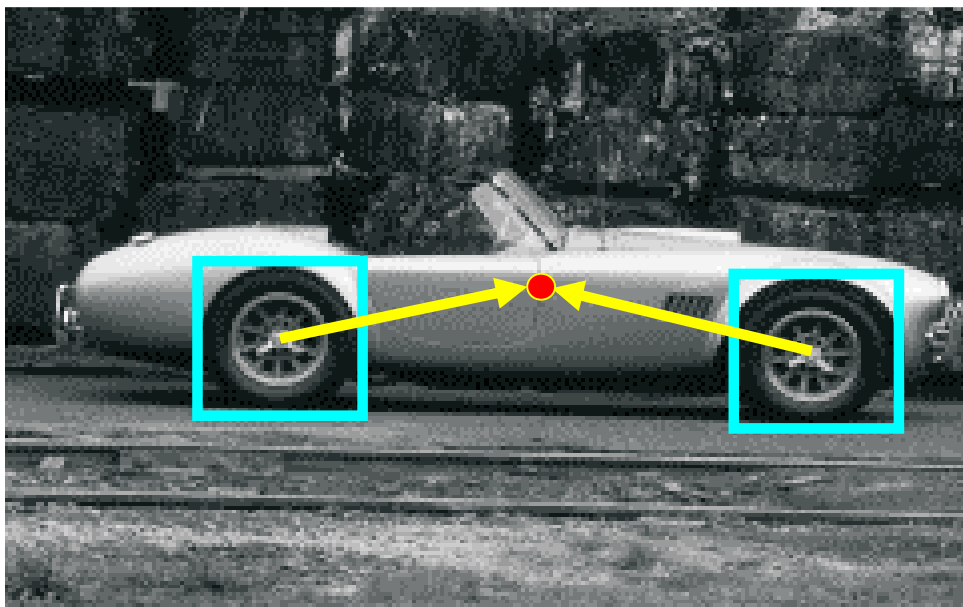
	 ...
	 ...
$\vdots$	

*Predpostavka: edina transformacija je translacija – torej, orientacija in skaliranje sta fiksirani.*

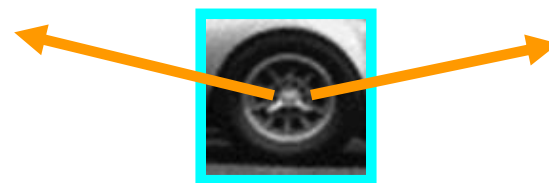


# Detekcija predmetov z GHT

- Vektorje odmika indeksiraj z lokalnim izgledom zaplat v sliki.



Učni primer



“vizualna beseda” z  
vektorji odmika

B. Leibe, A. Leonardis, and B. Schiele, Combined Object Categorization and Segmentation with an Implicit Shape Model, ECCV Workshop on Statistical Learning in Computer Vision 2004

Slide credit: S. Lazebnik

# Detekcija predmetov z GHT

- Vektorje odmika indeksiraj z lokalnim izgledom zaplat v sliki (vizualnih besed).



testna slika

B. Leibe, A. Leonardis, and B. Schiele, Combined Object Categorization and Segmentation with an Implicit Shape Model, ECCV Workshop on Statistical Learning in Computer Vision 2004

Slide credit: S. Lazebnik