Technical University of Cluj - Napoca
Computer Science Department

Sisteme de viziune in robotica

An2, Master Robotica

Cuprins / contents

Prelucrari pe imagini binare / *Binary image processing*

1. Determinarea componentelor conexe / etichetare. *(Labeling connected components)*
2. Detectia si urmarirea conturului. *(Countour tracing)*
3. Calculul proprietatilor geometrice ale obiectelor binare *(Simple geometric features of binary objects)*
4. Operatii morfologice si aplicatii *(Morphological operations and applications)*

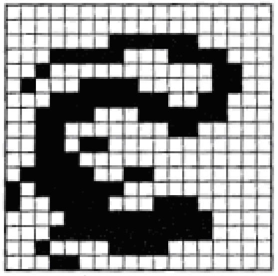
Imagine binara (binary image) ?

Images that contain only 2 colors / labels:

“0” – background pixels

“1” – object pixels

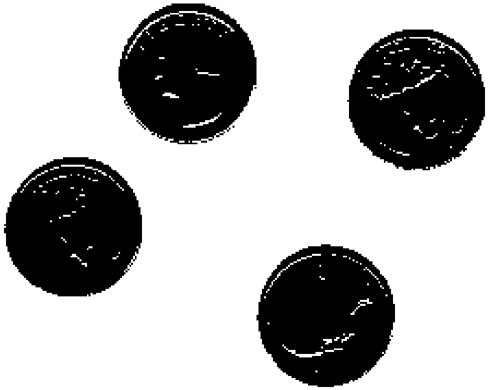
Obtained after a segmentation process !



Grayscale image

⇒

Segmentation



Binary image

Technical University of Cluj Napoca
Computer Science Department

Etichetarea obiectelor

din imagini binare

Labeling connected components

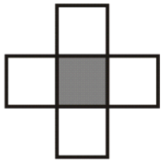
Technical University of Cluj Napoca
Computer Science Department

Definitions

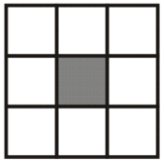
1. Vecini (Neighbors)

- 2 pixels are in a neighborhood relation N4 if they have a common frontier
- 2 pixels are in a neighborhood relation N8 if they have at least a common corner

N4



N8



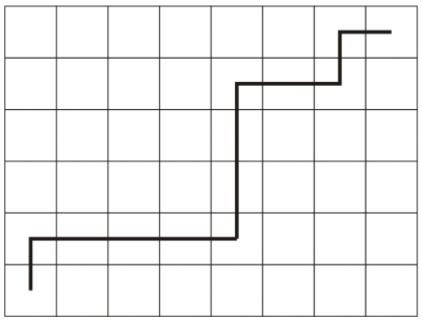
2. Cale (Path)

$$\text{Path}\ (p\ [i_0, j_0] \Rightarrow p\ [i_n,j_n]\ ) := \{\ [i_0,j_0], [i_1,j_1], \ldots, [i_n,j_n] \ \mid\ [i_k,j_k]\ N_{4/8}\ [i_{k+1},j_{k+1}]\ \forall\ k = 0\ ..\ n-1\ \}$$

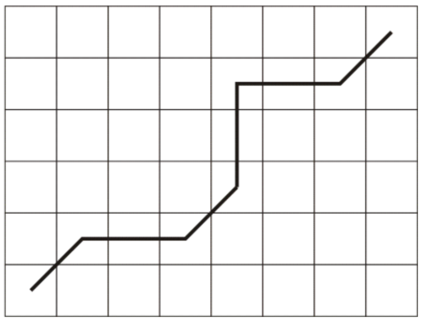
N4 $\Rightarrow$ 4-path

N8 $\Rightarrow$ 8-path

4-path



8-path



Definitions

3. Obiect (Foreground)

$S := \{ p[i,j] \mid p[i,j] = \text{"1"} \}$

4. Conectivitate (Connectivity)

$p_S \leftrightarrow q_S$ (connected) if $\exists$ Path $(p \Rightarrow q) \subset S$.

5. Componente conexe (Connected components)

$\{p_i \in S, i = 1 \dots n \mid p_k \leftrightarrow p_j, \forall (p_k, p_j) \in S, k,j = 1 \dots n\}$

6. Fundal (Background) := set of all connected components of C(S) which have points on the image margins. All other connected components from C(S) are *holes*.

7. Frontiera/Margine (Boundary)

Boundary (S): $= S' = \{ p \in S \mid \exists q \in N_{4/8}(p), q \in C(S) \}$

C(S) – complement of S

8. Interior

Interior (S) = S - S'

SVR

Connected component (*object*)

Maximal set of connected components:

$\{p_i \in S\ ,\ i = 1 \dots n \mid p_k \leftrightarrow pj,\ \forall\ (p_k, p_j) \in S,\ \ k,j = 1 \dots n\}$

A modality to label objects from a binary image is to chose a start point b_{ij} =1 and assign a label to the point and to its neighbors. Further the neighbors of the neighbors are labeled …..

- When the recursive procedure is finished, a connected component is obtained and we can continue by choosing another start point (not labeled yet).
- To find this new start point, the image is scanned systematically and a new labeling procedure is initiated when an object point b_{ij} =1 is found.

$\Rightarrow$

Etichetare (Labeling)

Sequential labeling

Iterative algorithm (Haralick 1981)

- No need for extra memory (memory efficient).
- Processing time depends on the image size/complexity.

1. Initialization phase
2. Repeat

        propagate labels top-down & left-right

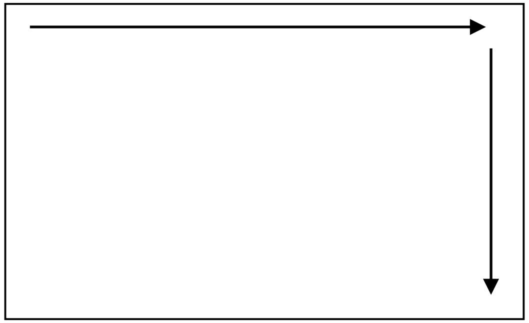
        propagate labels bottom-up & right-left

    until “no change”

```
procedure Iterate;
// Initialize each object pixel “1”with a unique label
for L:=1 to NLINES do
        for P:=1 to NCOLUMNS do
                if  I(L,P) =1
                then LABEL(L,P):=NEWLABEL()
                else LABEL(L,P):=0
        end for
end for;
```

Sequential labeling

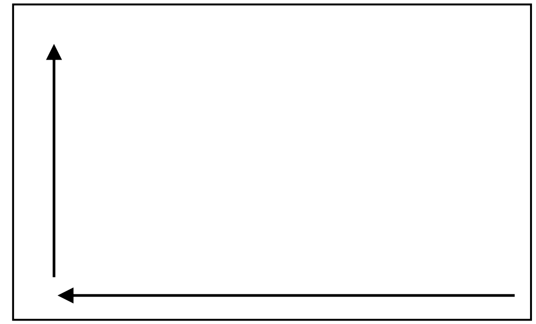
“**procedure** Iterate – pag. 2”
“Successive: top-down & bottom-up iterations”



```
repeat

CHANGE:=false;

// top-down iteration
for L:=1 to NLINES do
        for P:=1 to NCOLUMNS do
                if  LABEL(L,P)<>0 then
                        begin
                                M:=MIN(LABELS(NEIGHBORS(L,P)U(L,P)));
                                if M <> LABEL(L,P)
                                then CHANGE:=true;
                                LABEL(L,P):=M
                        end
        end for
end for;
```

Sequential labeling

“**procedure** Iterate – pag. 3”



```
// bottom-up iteration
 for L:= NLINES to 1 by –1 do
          for P:= NCOLUMNS to 1 by –1 do
                    if  LABEL(L,P)<>0 then
                              begin
                                        M:=MIN(LABELS(NEIGHBORS(L,P)U(L,P)));
                                        if M<> LABEL(L,P)
                                        then CHANGE:=true;
                                        LABEL(L,P):=M
                              end
          end for
end for;

until CHANGE:=false

end Iterate
```

Sequential labeling

Example (N4)

	1	1		1	1	
	1	1		1	1	
	1	1	1	1	1	

1. Initial image

	1	2		3	4	
	5	6		7	8	
	9	10	11	12	13	

2. Initialization

	1	1		3	3	
	1	1		3	3	
	1	1	1	1	1	

3. Top-down & left-right label propagation

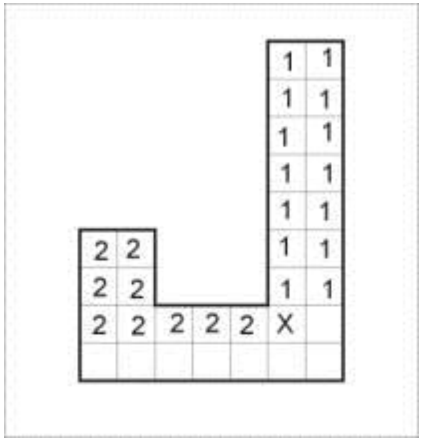
	1	1		1	1	
	1	1		1	1	
	1	1	1	1	1	

4. Bottom-up & right-left label propagation

Classic Algorithm (equivalence classes)

- Based on the classic algorithm that finds connected components in graphs
- Needs 2 iterations but a large table for the equivalences might be needed

1. 1-st step: labels propagation (similar with the previous algorithm)
   - When 2 different labels can be propagated to the same pixel, the smallest one is propagated and the found equivalence is stored in an equivalence table (ex. (1,2) $\rightarrow$ EqTable).
   - Every entry in the EqTable is an ordered pair containing the equivalent labels
   - After this step the equivalence classes are found
   - For every equivalence class a unique label is assigned (smallest or oldest value)



2. 2-nd step: the image is scanned and the corresponding label of the equivalence class is assigned to each pixel

Technical University of Cluj Napoca
Computer Science Department

Classic Algorithm

Example (N4)

1					1	1
		1	1			1
		1				1
		1				1
1	1	1				1
	1	1		1		1
	1	1	1	1		1
				1	1	1

1. Initial image

1					2	2
		3	3			2
		3				2
		3				2
4	4	3				2
	4	3		5		2
	4	3	3	3		2
				3	3	2

2. Label image after top-down propagation

EQTABLE:
(4, 3), (3, 5), (3, 2) …

EQCLASSES:
1: {4, 3, 5, 2}
2: (6,8,9, …}
….
n: {….}

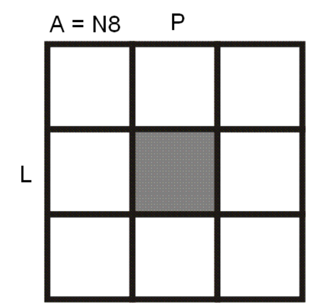
EQLABEL:

1: 2	1:1
2: 6 sau	2:2
…	…
n: x	n:n

SVR

Classic Algorithm

```
procedure Classical
"Initialize global equivalence table" and labels matrix
EQTABLE:=CREATE(); LABEL:=CREATE();

"Top-down pass 1"
for L:= 1 to NLINES do
        "Initialize all labels on line L to zero"
        for P:= 1 to NCOLUMNS do
                LABEL(L,P):=0
        end for
        "Process the line"
        for P:=1 to NCOLUMNS do
                if I(L,P):= 1 then
                        begin
                                A:= NEIGHBORS((L,P));
                                if ISEMPTY(A)
                                then M:=NEWLABEL()
                                else M:= MIN(LABELS(A));
                                LABEL(L,P):=M;
                                for X in LABELS(A) and X<>M
                                        ADD(X, M, EQTABLE)
                                end for;
                        end
        end for
end for;
```

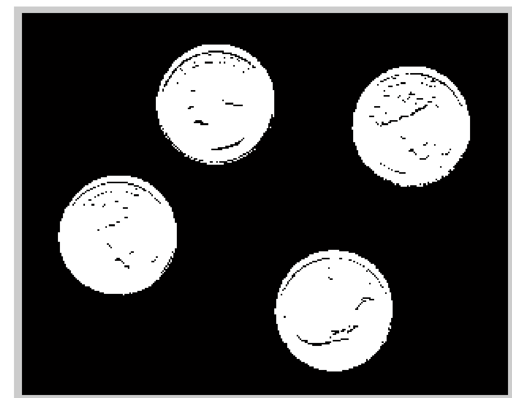


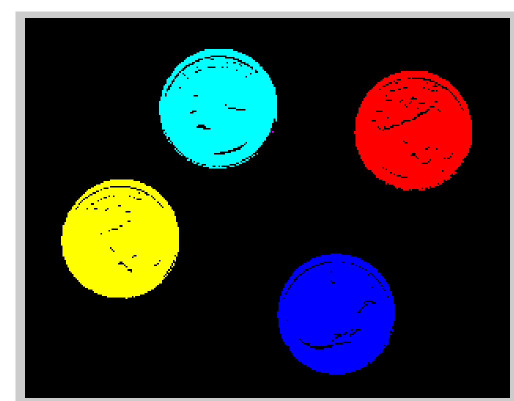
Classic Algorithm

```
"Find equivalence classes"
EQCLASSES:=Resolve(EQTABLE);

for E in EQCLASSES
        EQLABEL(E):= min(LABELS(E))
end for;

"Top-down pass 2"
for L:= 1 to NLINES do
        for P:= 1 to NCOLUMNS do
                if I(L,P) = 1
                then LABEL(L,P):=EQLABEL(CLASS(LABEL(L,P)))
        end for
end for
end Classical
```

- Resolve() - algorithm that finds the connected components of the graph defined by the equivalences set (EQTABLE) defined at step 1.
- Problem: for big images with many objects the table is large (large memory usage)

Example





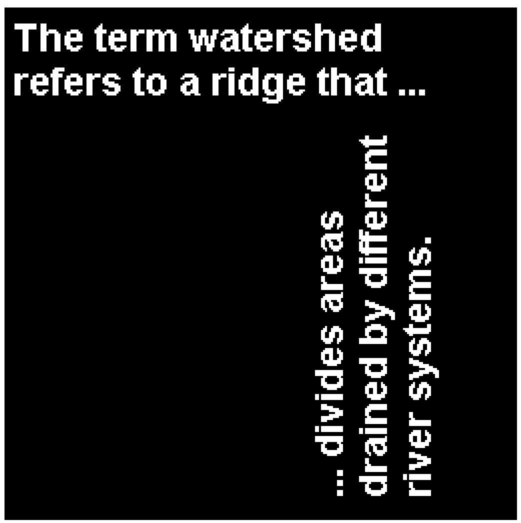
```
%read the grayscale image
I=imread('eight.bmp','BMP');
ColorDepth=256;
figure; imshow(I);
%Cimpute the threshold(see C2)
T=hist_threshold(I);
T_norm = T/ ColorDepth
%normalize the threshold: 0 ... 1
Ibw=im2bw(I, T_norm);
%Image negative:
%Background pixels: 0 (black)
%Object pixels: 1 (white)
Ibw=~Ibw;
figure; imshow(Ibw);
%Objects labeling
%Ilabel: matrix containing the labels
%0 - background, 1 - obiect1 label, 2 - obiect 2 label,
[Ilabel,num] = bwlabel(Ibw,8);
% Display the labels matrix in colors
Irgb= label2rgb(Ilabel, 'hsv', 'black', 'shuffle');
figure; imshow(Irgb);
```

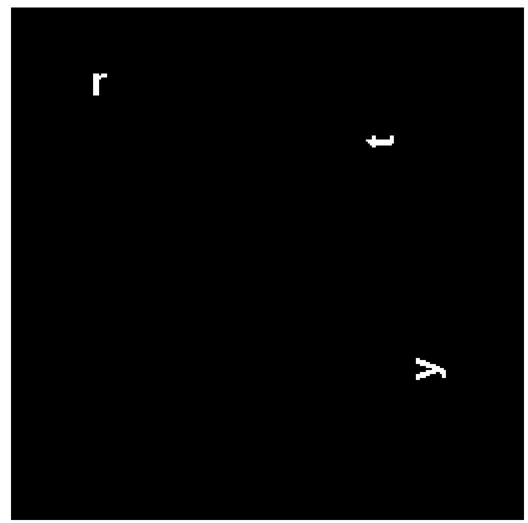
[L,NUM] = bwlabel(BW,N) - returns a matrix L, of the same size as BW, containing labels for the connected components in BW. N can have a value of either 4 or 8. 4 specifies N4 and 8 specifies N8.

Example

To select object from a binary image we can use the function *bwselect*, by specifying the coordinates of a pixel inside the object:

```
BW1 = imread('text.png');
c = [43 185 212];
r = [38 68 181];
BW2 = bwselect(BW1,c,r,4);
imshow(BW1), figure, imshow(BW2)
```





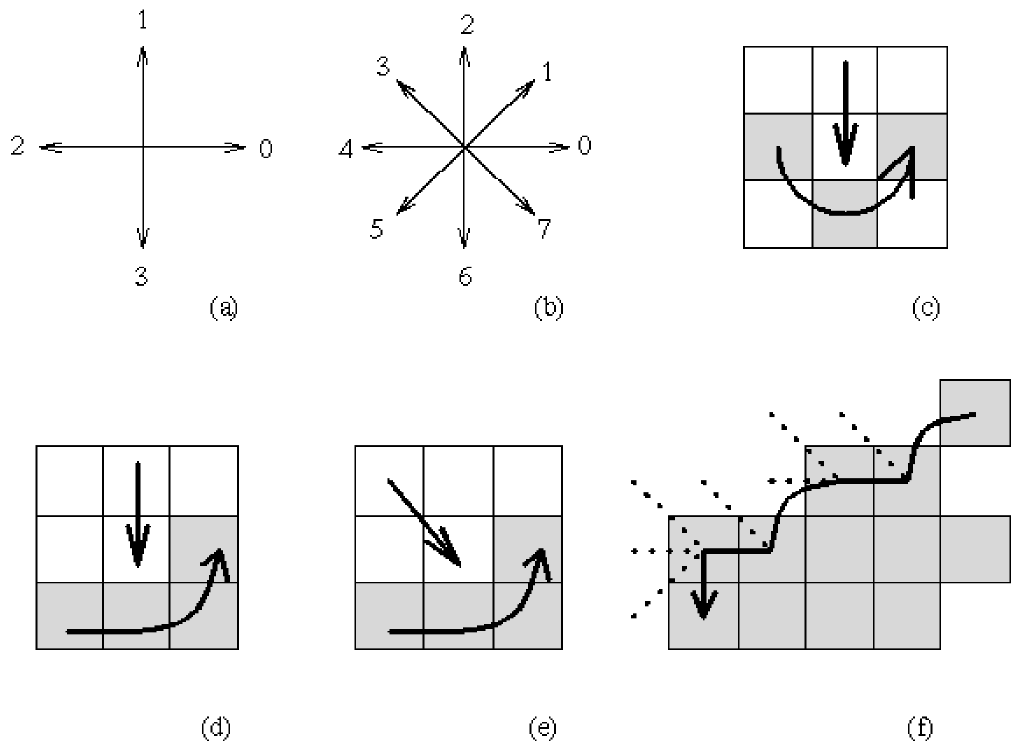
Contour Tracing (detectie contur)

Contour:

Contour(R) = { p ∈ R | ∃ q ∈ N4/8(p), q ∈ C(R) }

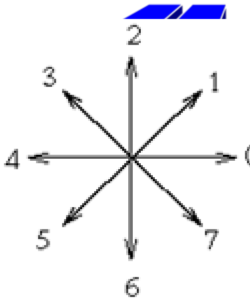
- *chain-code / direction codes*: *c*

(numerical operations applied on *c* are mod 4 or 8)



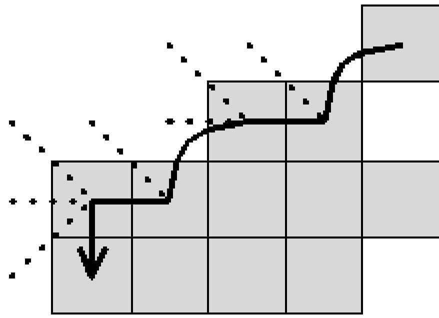
Technical University of Cluj Napoca
Computer Science Department

Contour Tracing (detectie contur)



Contour tracing algorithm:

1. Scan the image(top-down + left-right) until it finds a start pixel *P0. Define a* variable *dir* which stores the last movement direction along the contour (from the previous to the current point):
- *dir* = 0 for N4
- *dir* = 7 for N8



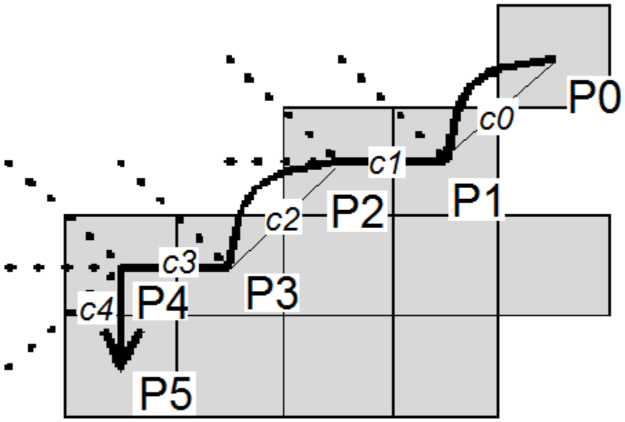
2. Search the next contour point in a neighborhood of 3x3 around the current pixel, by sequentially incrementing (dir++), In counterclockwise direction starting with direction:
- (*dir* + 3) *mod* 4 (N4)
- (*dir* + 7) *mod* 8 if *dir* is even (N8)
- (*dir* + 6) *mod* 8 if *dir* is odd (N8)

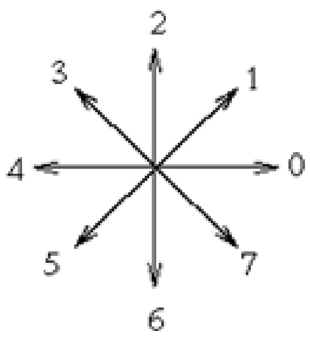
First pixel of "1" is the current contour pixel: *Pn*. In the same time it updateds *dir*.

3. If the current contour element *Pn* is identical with *P1* and if element *Pn-1* is identical with *P0*, STOP. Otherwise repeat step 2.

4. The detected contour is: *P0* … *Pn-2*.

Example – contour representation





$c_i \in \{0,1, \ldots ,7\}$ – direction codes

Var.1 – list of points:

$L = \{ P_0(x_0,y_0), P_1(x_1,y_1), \ldots , P_{n-2}(x_{n-2},y_{n-2}) \}$

Var .2 – chain codes:

$P_0(x_0,y_0) + \{c_0, c_1, \ldots , c_{n-2}\}$,

Var. 3 – *chain codes derivative* (invariant to rotation)

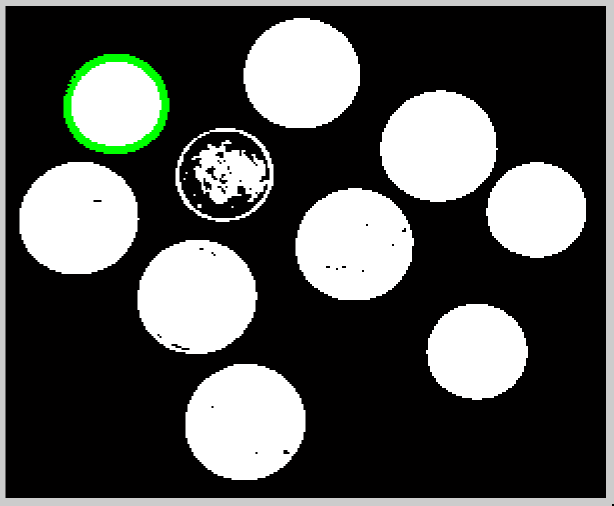
$P_0(x_0,y_0) + \{cd_0, cd_1, \ldots , cd_{n-2}\}$,

where: $cd_i = (c_i - c_{i-1})$ *mod* 8, $cd_0 = c_0$ *mod* 8

Technical University of Cluj Napoca
Computer Science Department

Example

```
I = imread('coins.png');
figure; imshow(I)
%thresholding / image segmentation
BW = im2bw(I);

%select a start point
dim = size(BW)
col = round(dim(2)/2)-90;
row = min(find(BW(:,col)))

boundary = bwtraceboundary(BW,[row, col],'N');

%displat the BW image and the contour
figure; imshow(BW)
hold on;
plot(boundary(:,2),boundary(:,1),'g','LineWidth',3);
```



SVR