Digital Image Processing

Lecture # 13
Image Segmentation & Hough Transform

Image Segmentation

Image Segmentation

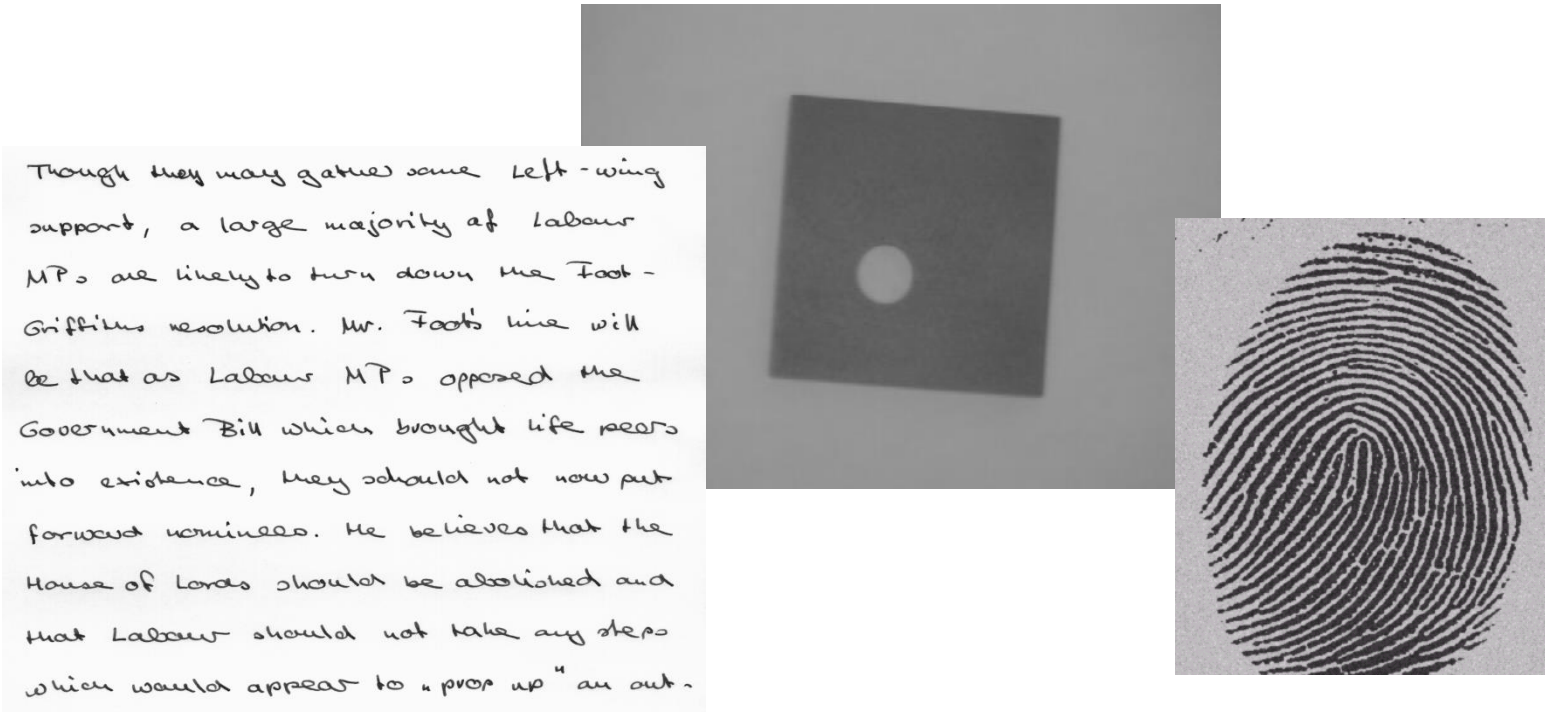
- Group similar components (such as, pixels in an image, image frames in a video)
- Applications: Finding tumors, veins, etc. in medical images, finding targets in satellite/aerial images, finding people in surveillance images, summarizing video, etc.

Image Segmentation

- Segmentation algorithms are based on one of two basic properties of gray-scale values:
  - Discontinuity
    - Partition an image based on abrupt changes in gray-scale levels.
    - Detection of isolated points, lines, and edges in an image.
  - Similarity
    - Thresholding, region growing, and region splitting/merging.

Thresholding

- Segmentation into two classes/groups
  - Foreground (Objects)
  - Background



Thresholding

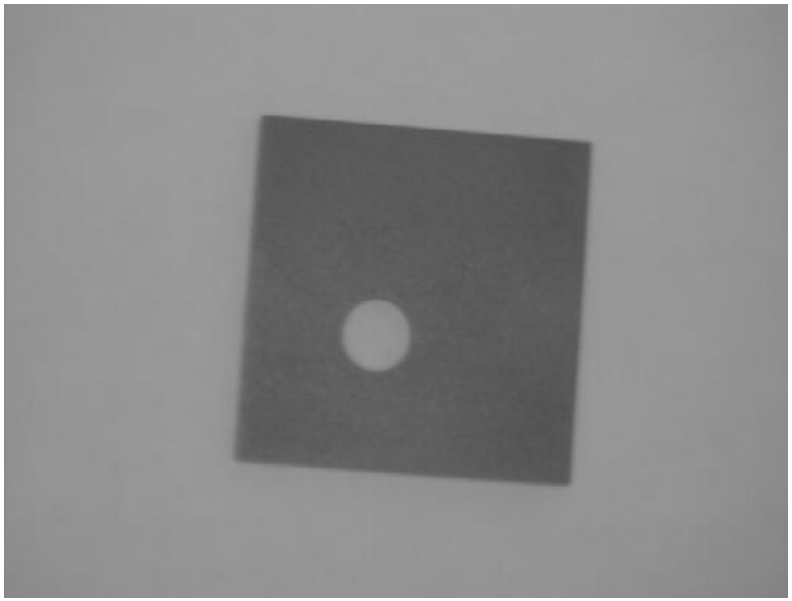
$$g(x,y) = \begin{cases} 1 & if \;\; f(x,y) > T \\ 0 & if \;\; f(x,y) \leq T \end{cases}$$

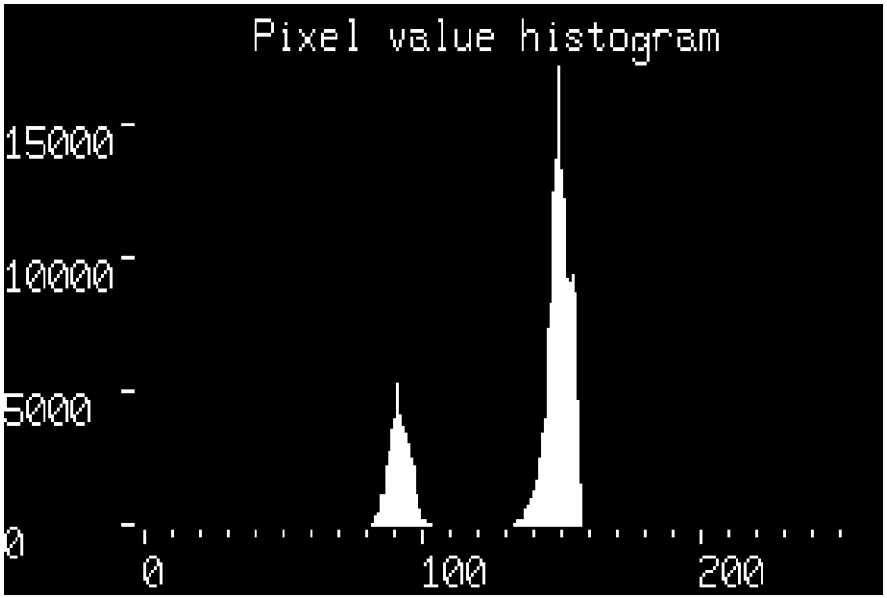
Objects & Background

- Global Thresholding
- Local/Adaptive Thresholding

Global Thresholding

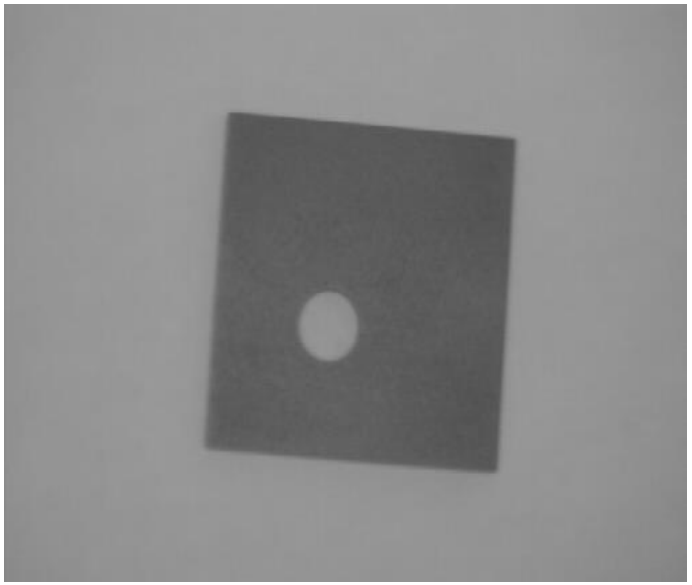
- Single threshold value for entire image
- Fixed ?
- Automatic
  - Intensity histogram

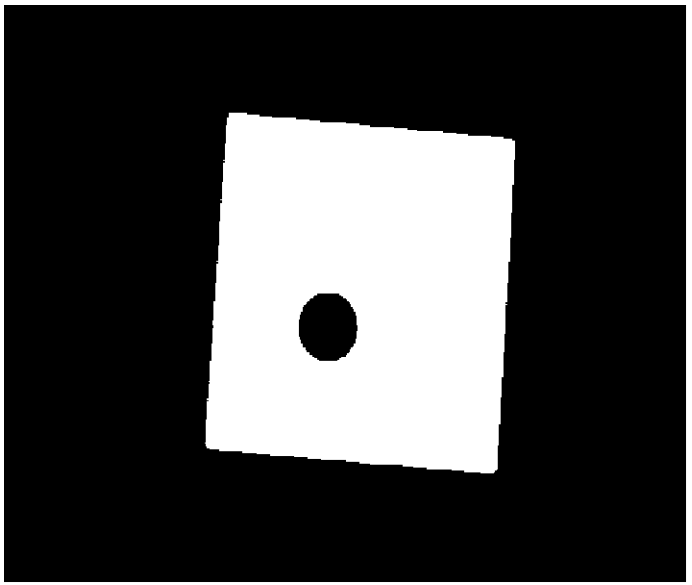




Global Thresholding

- Single threshold value for entire image
- Fixed ?
- Automatic
  - Intensity histogram

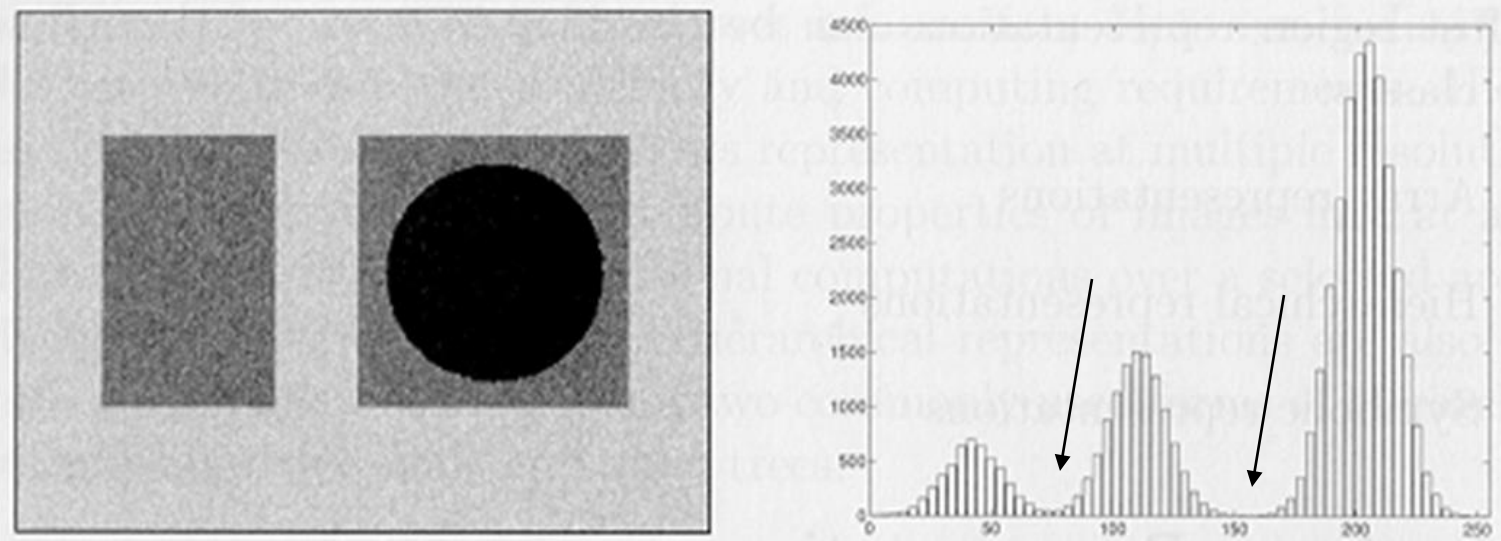




Global Thresholding

- Estimate an initial T
- Segment Image using T: Two groups of pixels G1 and G2
- Compute average gray values m1 and m2 of two groups
- Compute new threshold value T=1/2(m1+m2)
- Repeat steps 2 to 4 until: abs(T_i – T_{i-1})<epsilon

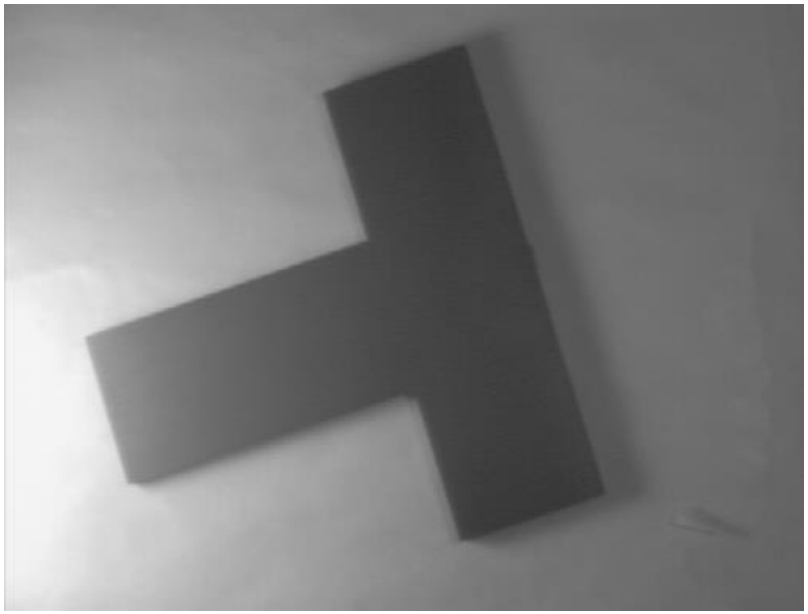
Global Thresholding

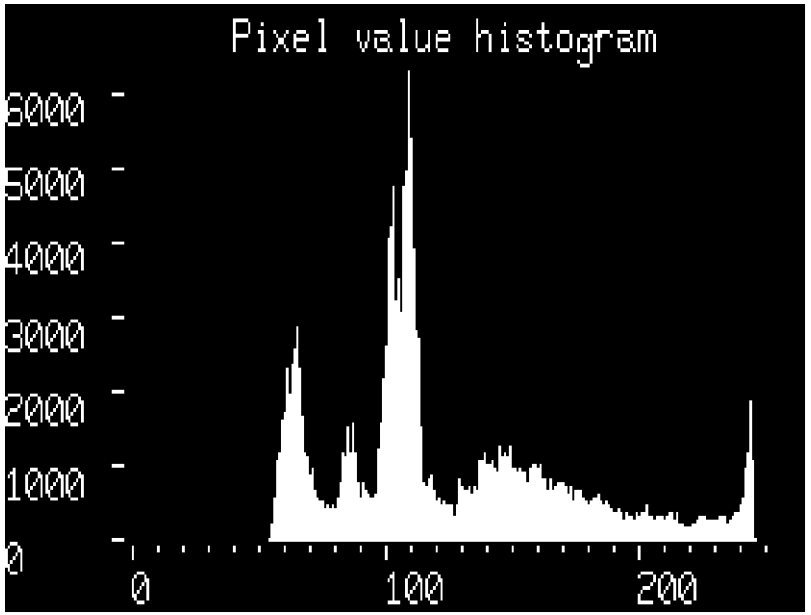


Multilevel thresholding

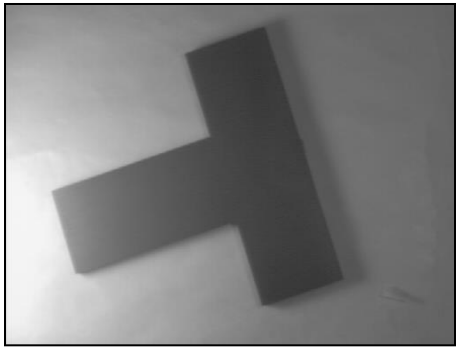
Thresholding

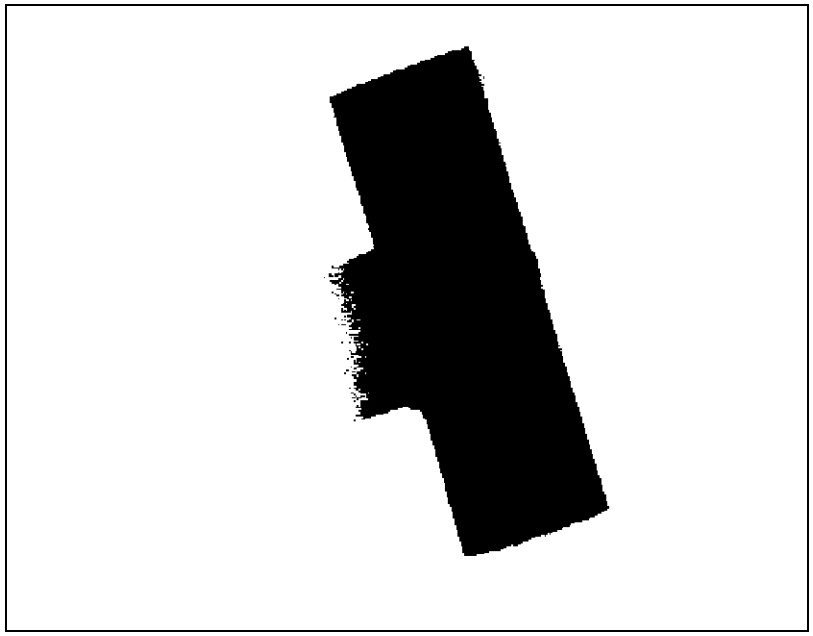
- Non-uniform illumination:

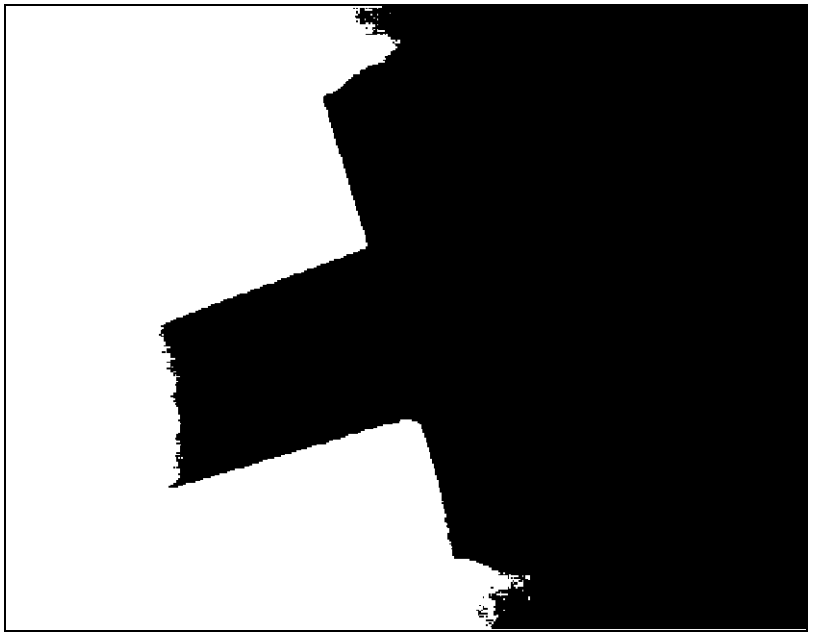




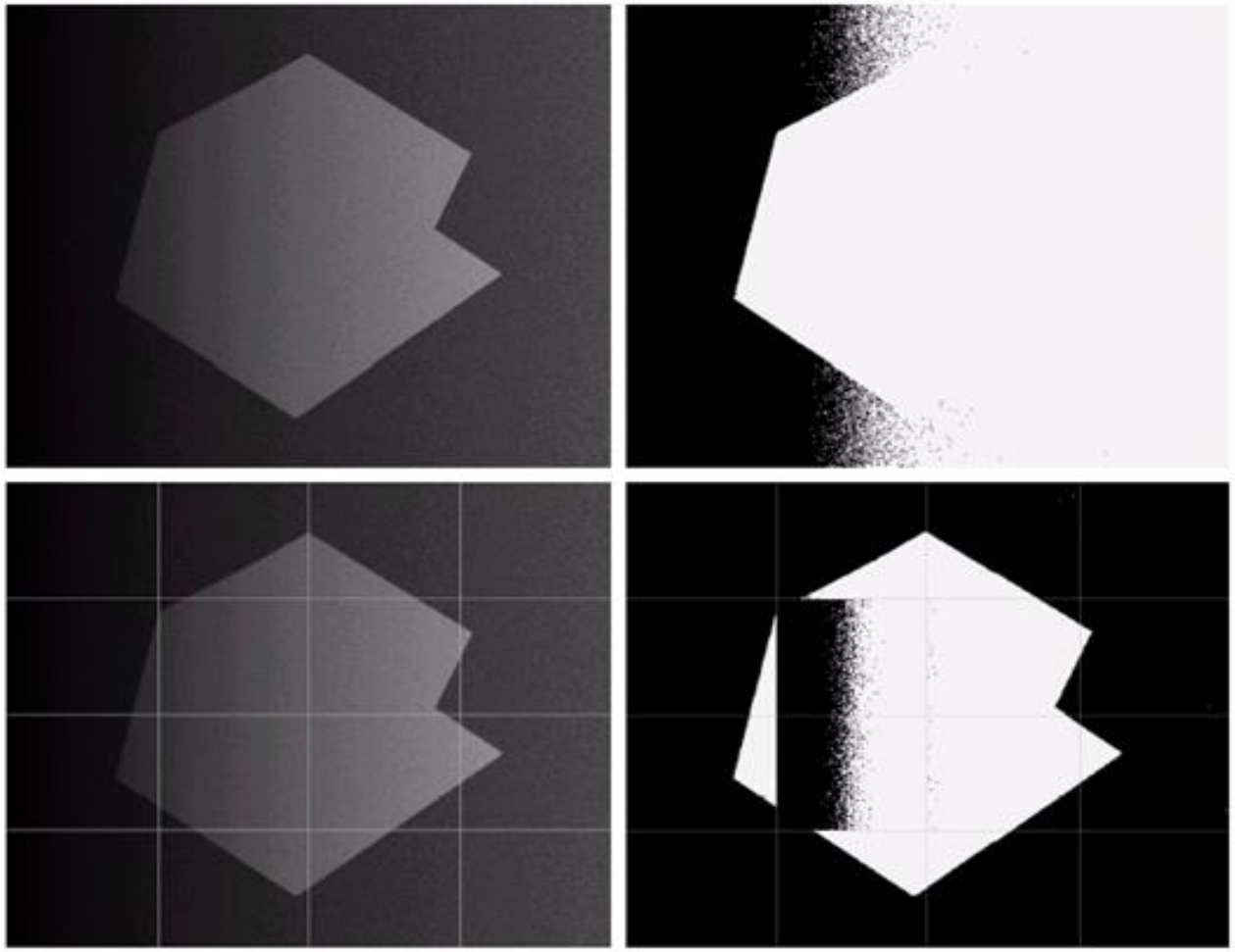
Global Thresholding







Adaptive Thresholding



Adaptive Thresholding

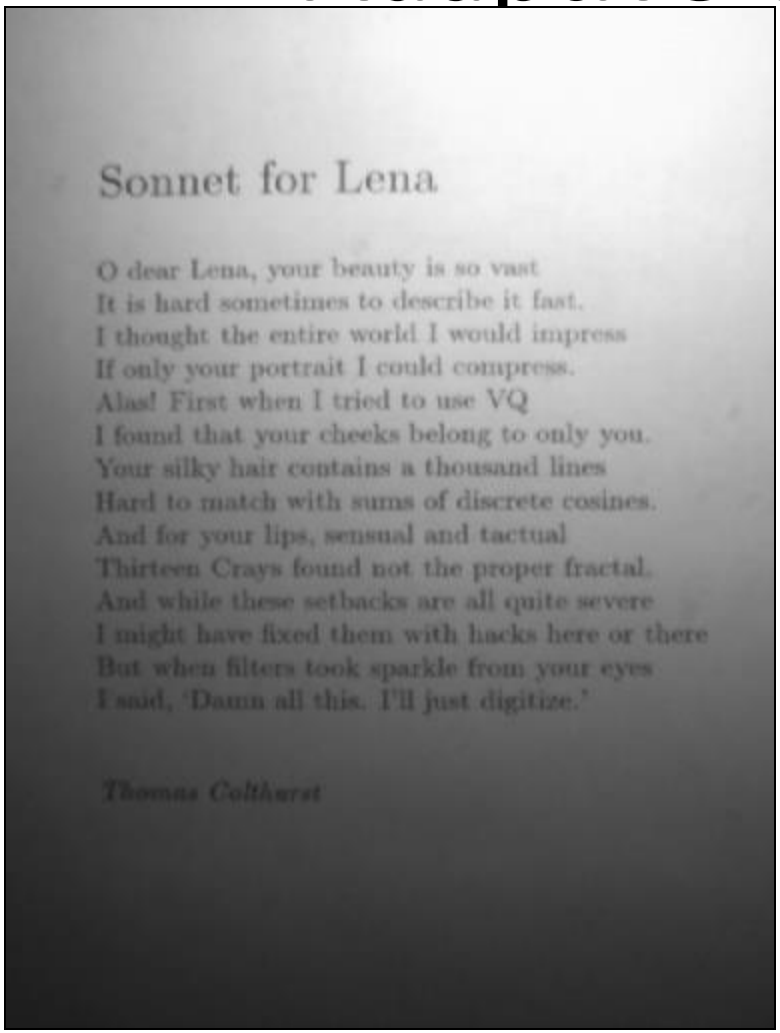
- Threshold: function of neighboring pixels

$$T = mean$$

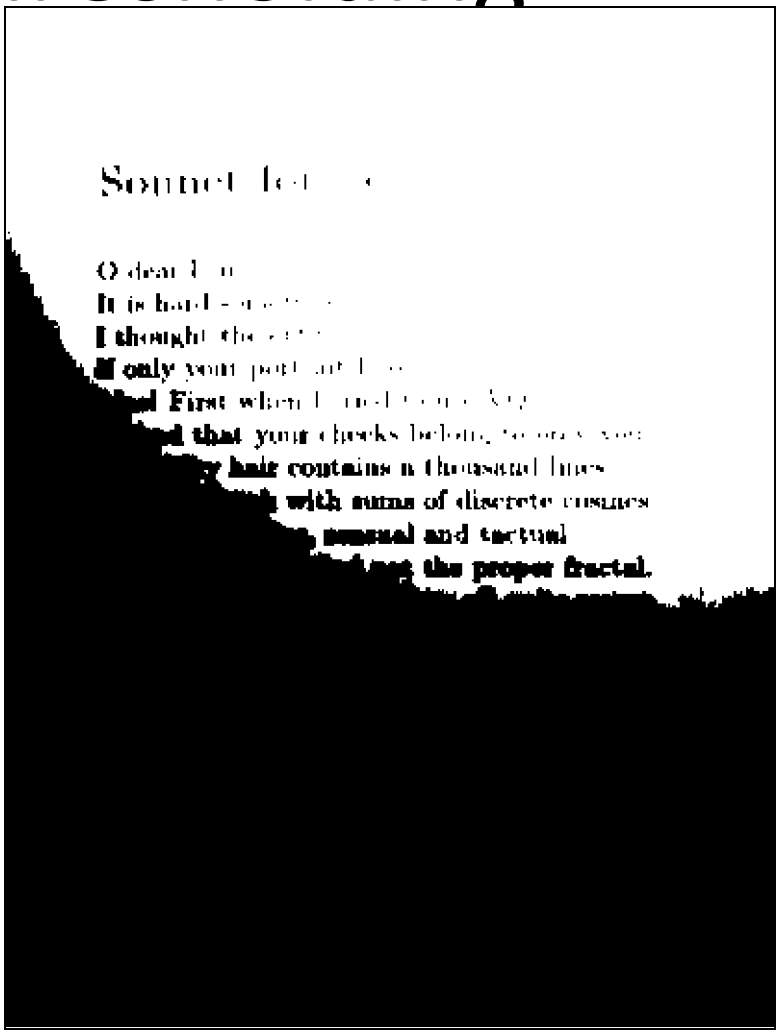
$$T = median$$

$$T = \frac{\max + \min}{2}$$

Adaptive Thresholding

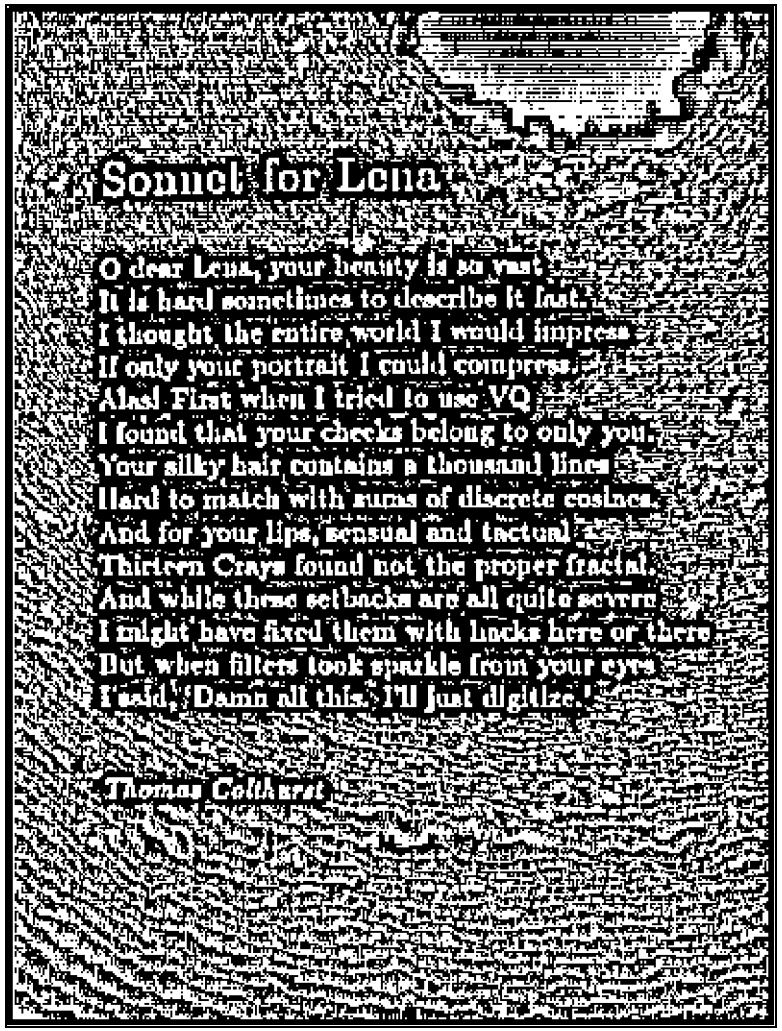


Original Image

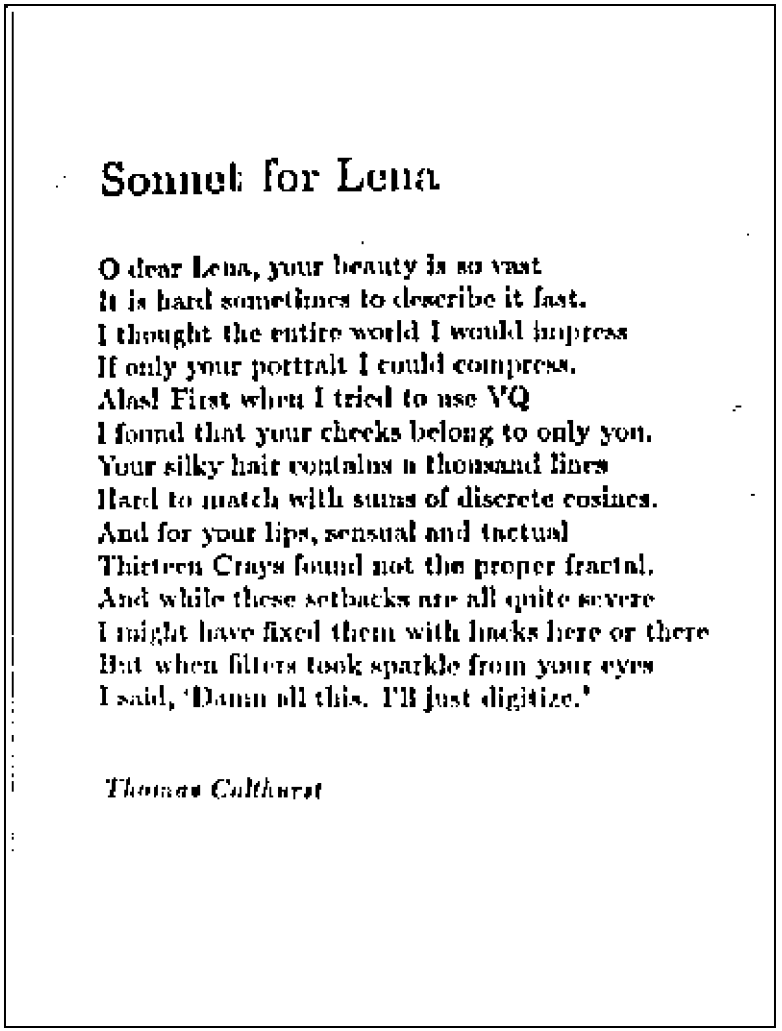


Global Thresholding

Adaptive Thresholding



T=mean, neighborhood=7x7



T=mean-Const., neighborhood=7x7

Adaptive Thresholding

- Niblack Algorithm

$$T = m + k \times s$$

$$m = \text{mean}$$

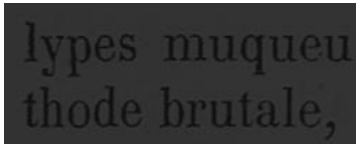
$$s = \text{standard deviations}$$

$$k = \text{ Niblack constant}$$

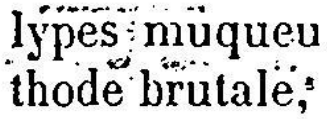
- Neighborhood size???

Document Binarization

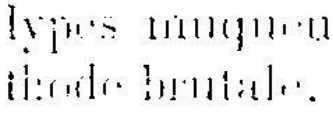
- Local Thresholding – Examples



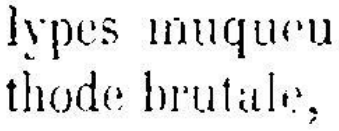
Original



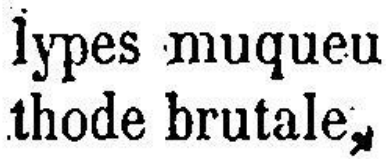
Niblack



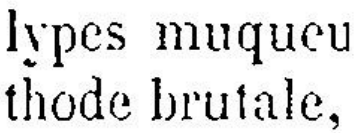
Sauvola



Wolf



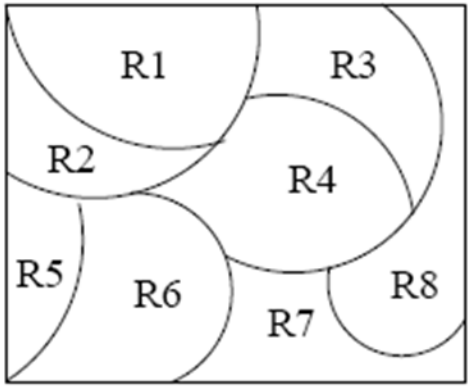
Feng



NICK

Region-Based Segmentation

- Divide the image into regions
  - $R_1, R_2, \ldots, R_N$
- Following properties must hold:



(1) $R_1 \cup R_2 \cup \cdots \cup R_n = R$

(2) R_i is connected

(3) $R_i \cap R_j = empty$

(4) $P(R_i) = \text{True}$

(5) $P(R_i \cup R_j) = \text{False}$ (For adjacent regions)

Region-Based Segmentation

- Region Growing
  - Region growing: groups pixels or subregions into larger regions.
  - *Pixel aggregation:* starts with a set of "seed" points and from these grows regions by appending to each seed points those neighboring pixels that have similar properties (such as gray level).

1. Choose the seed pixel(s).
2. Check the neighboring pixels and add them to the region if they are similar to the seed
3. Repeat step 2 for each of the newly added pixels; stop if no more pixels can be added

Predicate: for example abs(z_j - seed) < Epsilon

Region-Based Segmentation

- Example

10	10	10	10	10	10	10
10	10	10	69	70	10	10
59	10	60	64	59	56	60
10	59	10	60	70	10	62
10	60	59	65	67	10	65
10	10	10	10	10	10	10
10	10	10	10	10	10	10

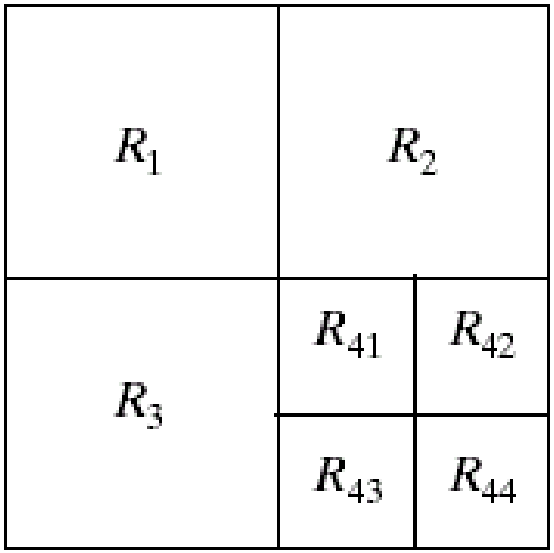
Region-Based Segmentation

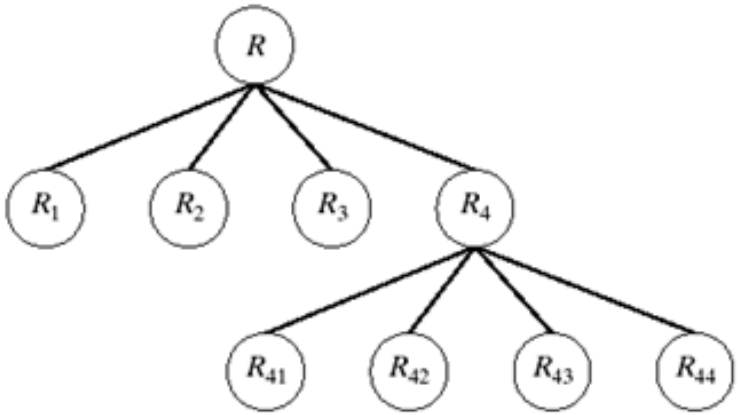
- Region Splitting
  - Region Growing: Starts from a set of seed points.
  - Region Splitting: Starts with the whole image as a single region and subdivide the regions that do not satisfy a condition.
  - Image = One Region R
  - Select a predicate P (gray values etc.)
  - Successively divide each region into smaller and smaller quadrant regions so that:

$$P(R_i) = true$$

Region-Based Segmentation

- Region Splitting





Problem? Adjacent regions could be same

Solution? Allow Merge

Region-Based Segmentation

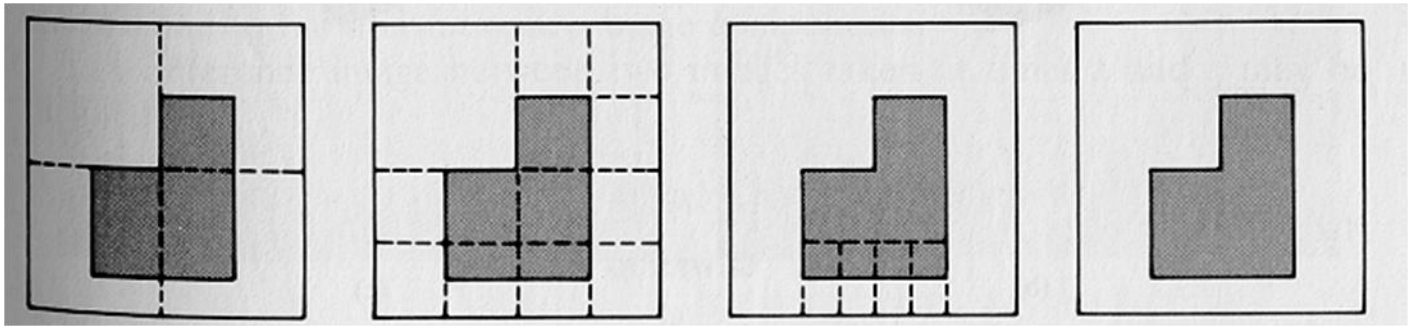
- Region Merging
  - Region merging is the opposite of region splitting.
  - Merge adjacent regions R_i and R_j for which:

$$P(R_i \cup R_j) = True$$

- Region Splitting/Merging
  - Stop when no further split or merge is possible

Region-Based Segmentation

- Example



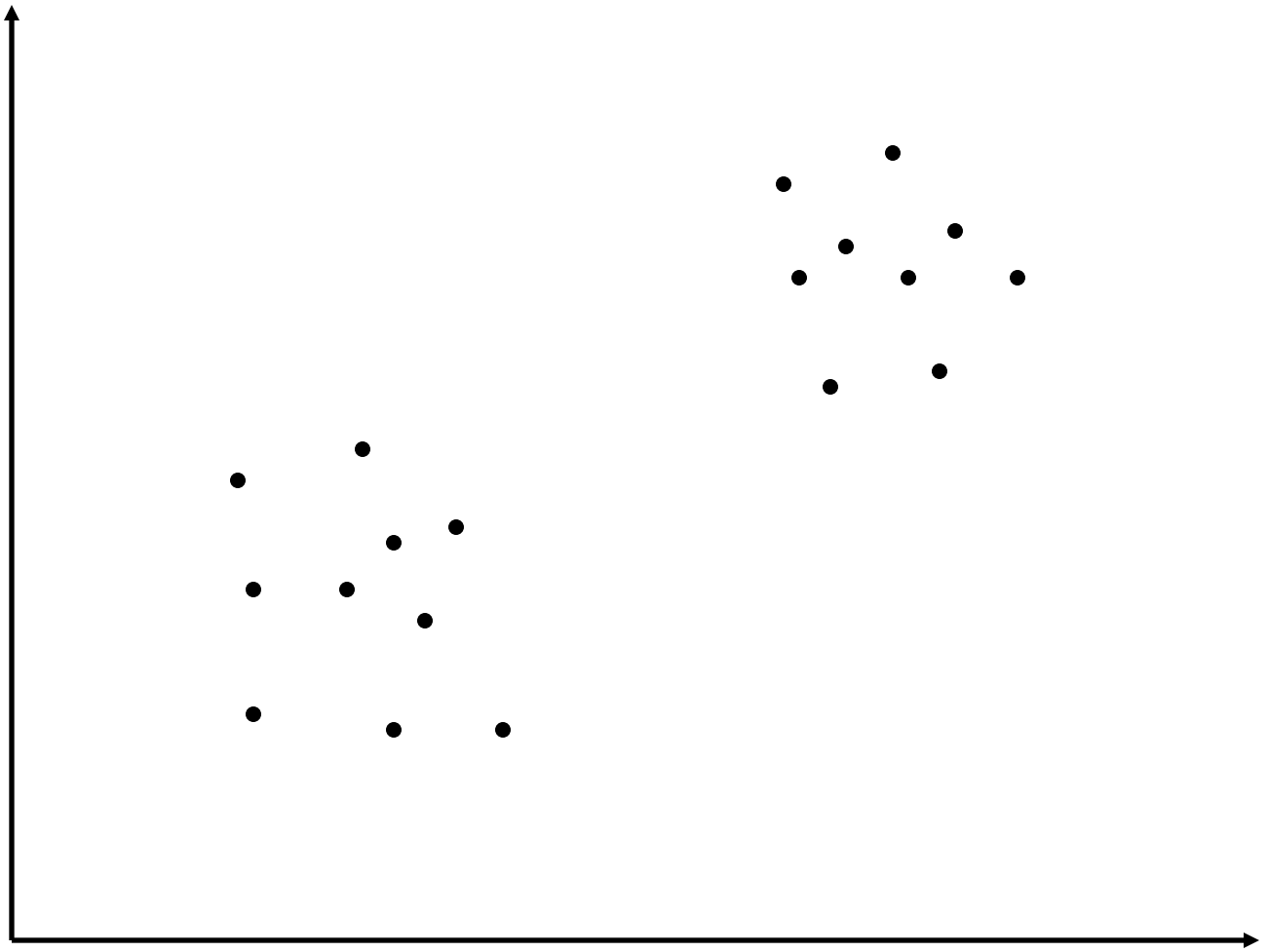
1. Split into four disjointed quadrants any region R_i where $P(R_i)$=False
2. Merge any adjacent regions R_j and R_k for which $P(R_j \cup R_k)$=True
3. Stop when no further merging or splitting is possible

Clustering

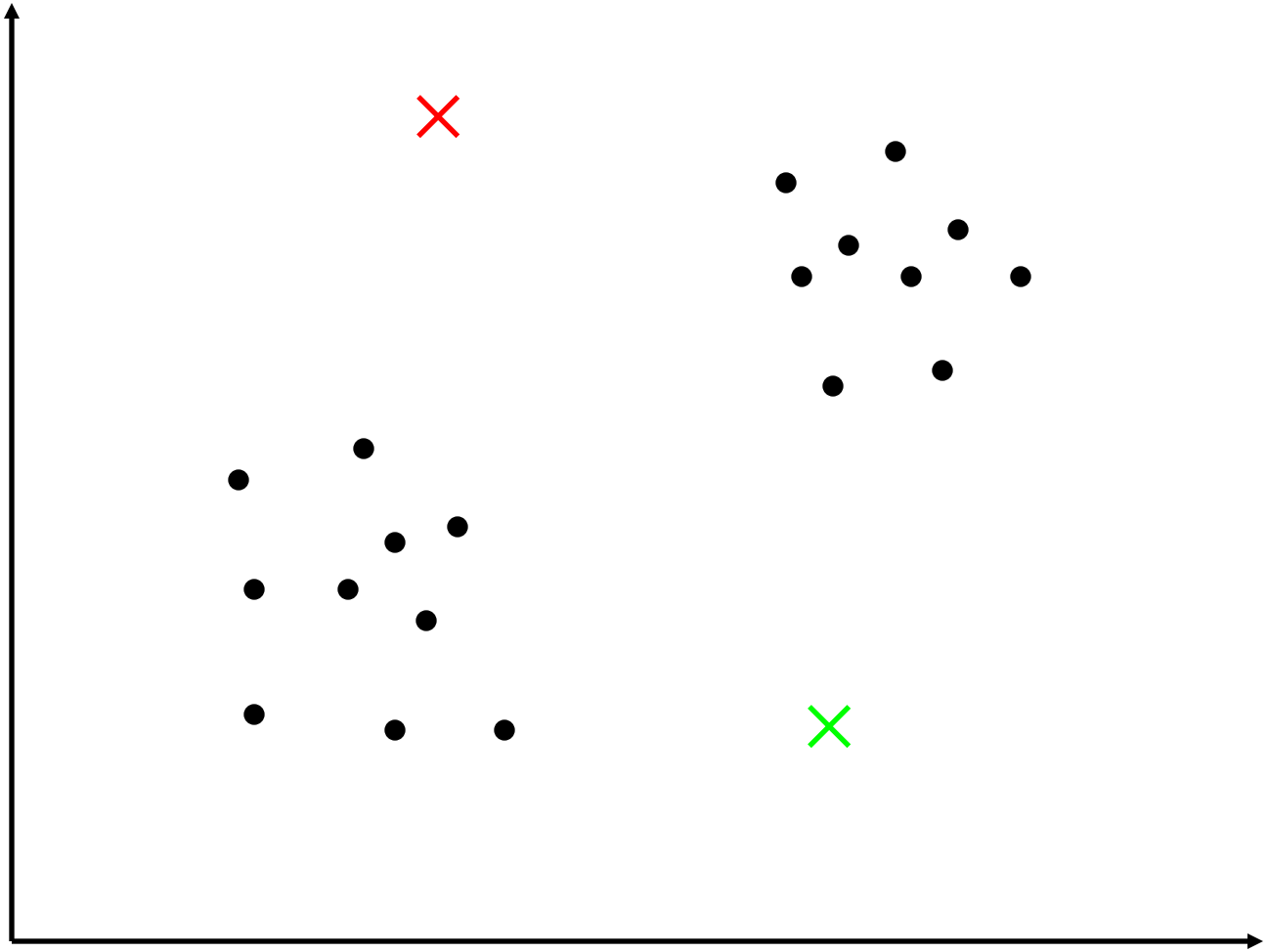
K-Means Clustering

1. Chose the number (K) of clusters and randomly select the centroids of each cluster.
2. For each data point:
   - Calculate the distance from the data point to each cluster.
   - Assign the data point to the closest cluster.
3. Recompute the centroid of each cluster.
4. Repeat steps 2 and 3 until there is no further change in the assignment of data points (or in the centroids).

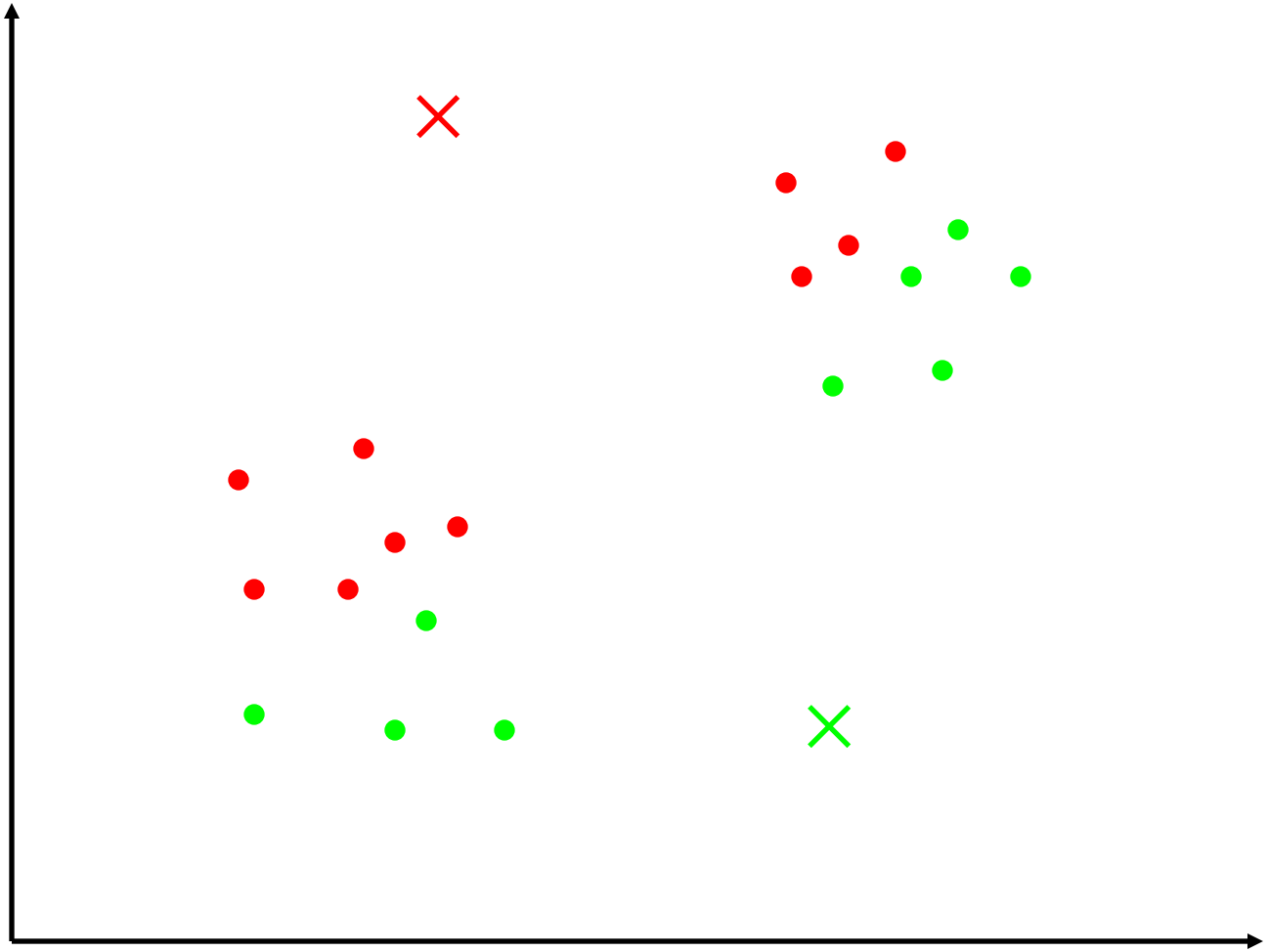
K-Means Clustering



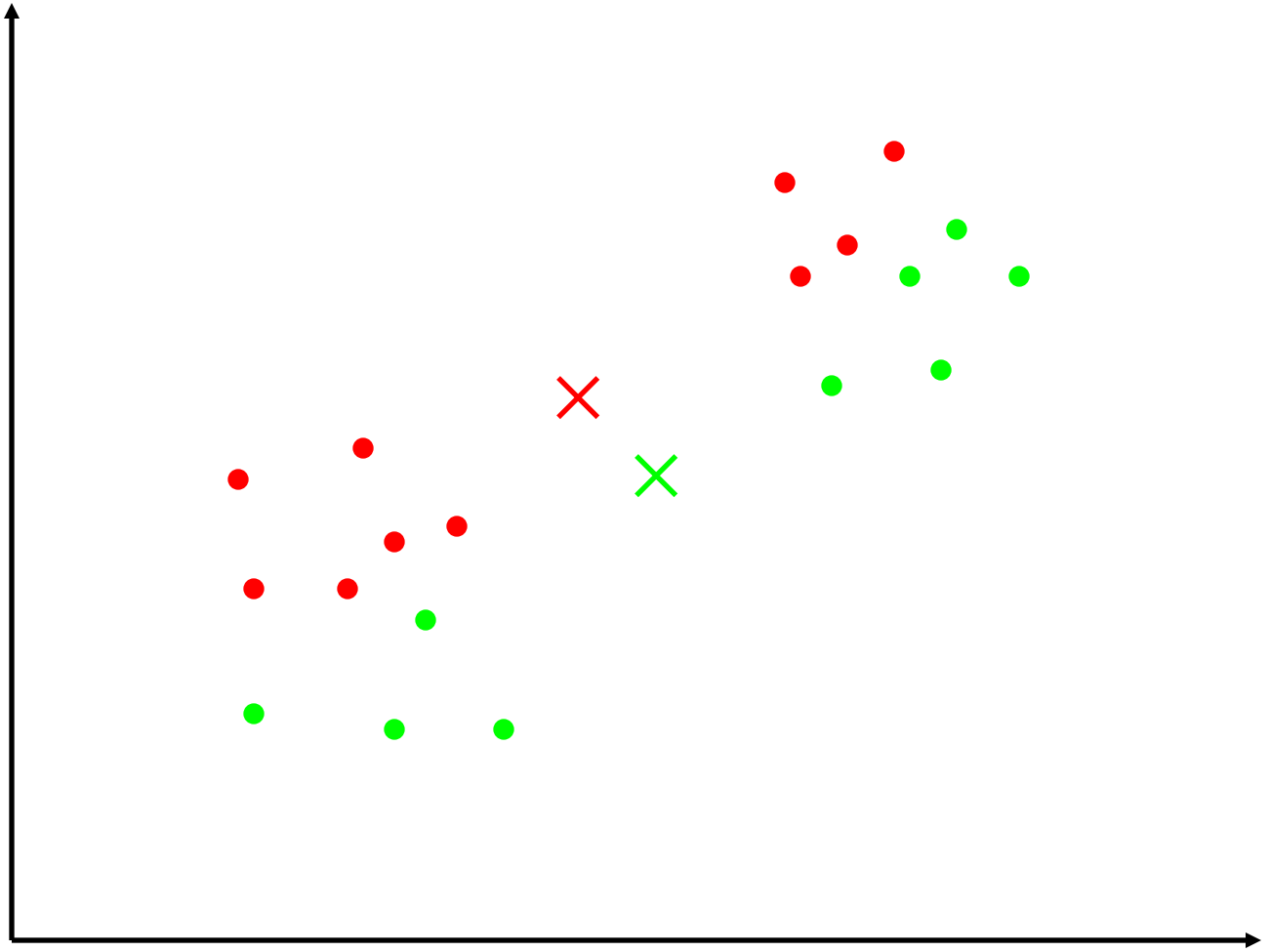
K-Means Clustering



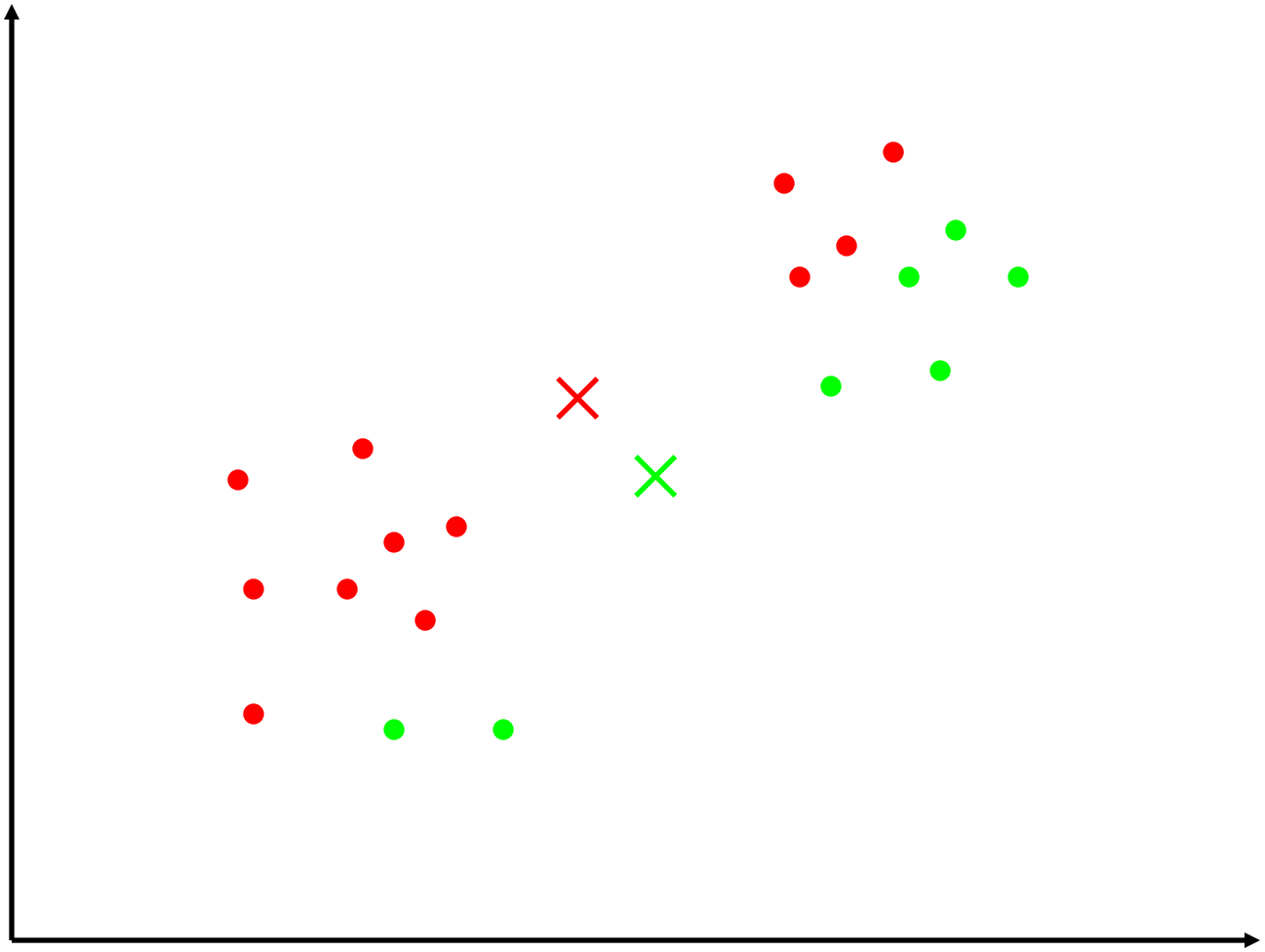
K-Means Clustering



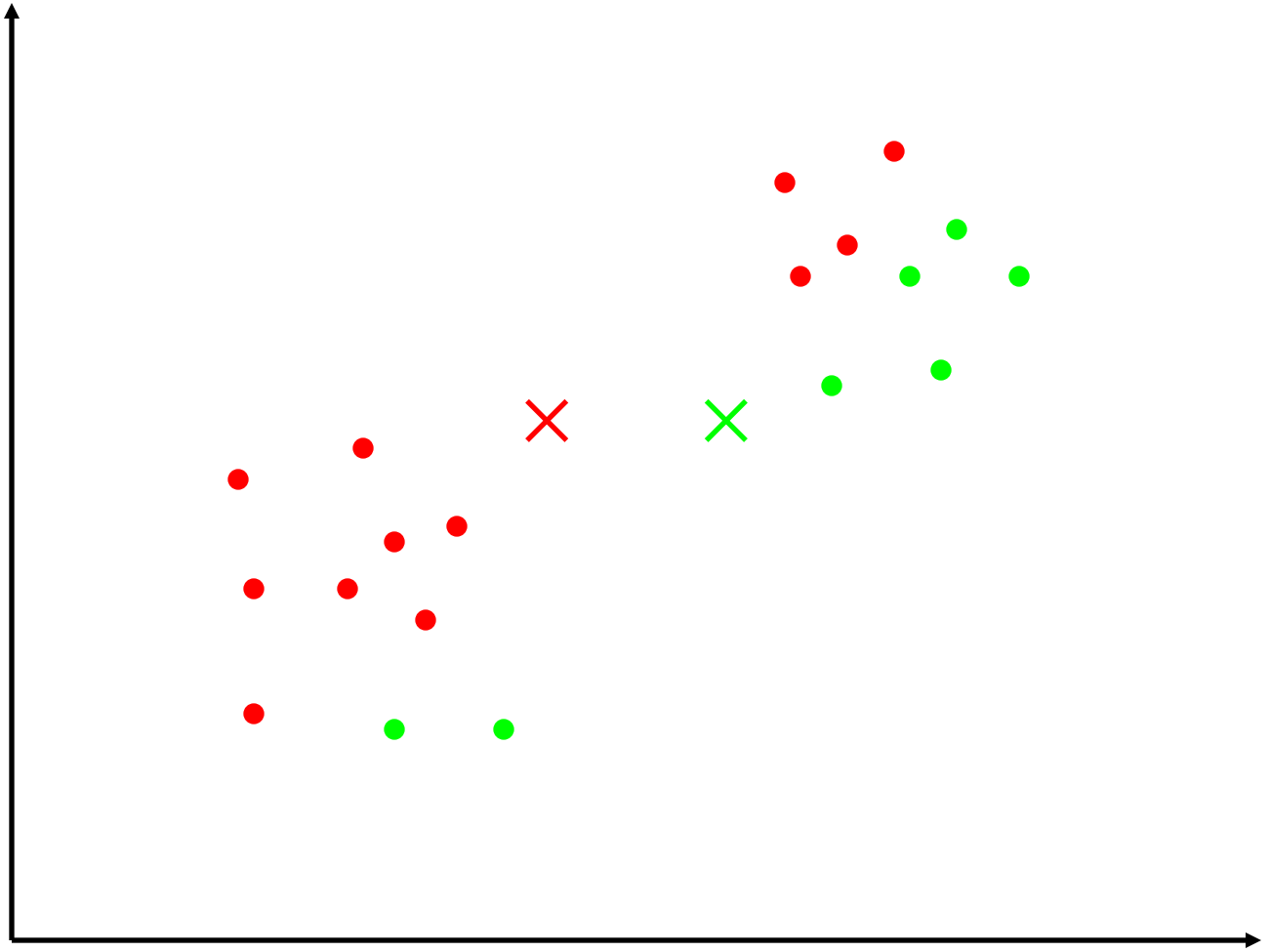
K-Means Clustering



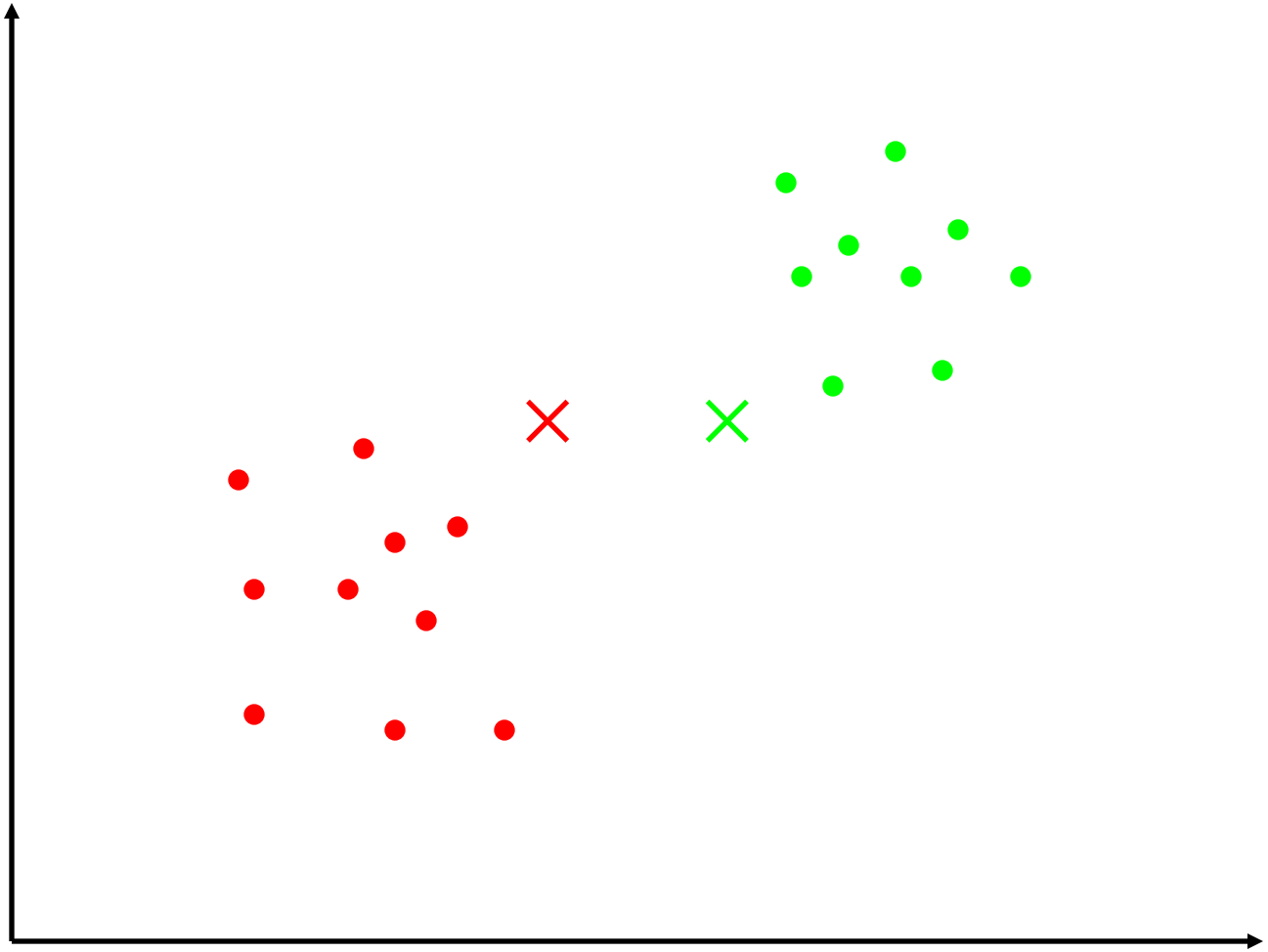
K-Means Clustering



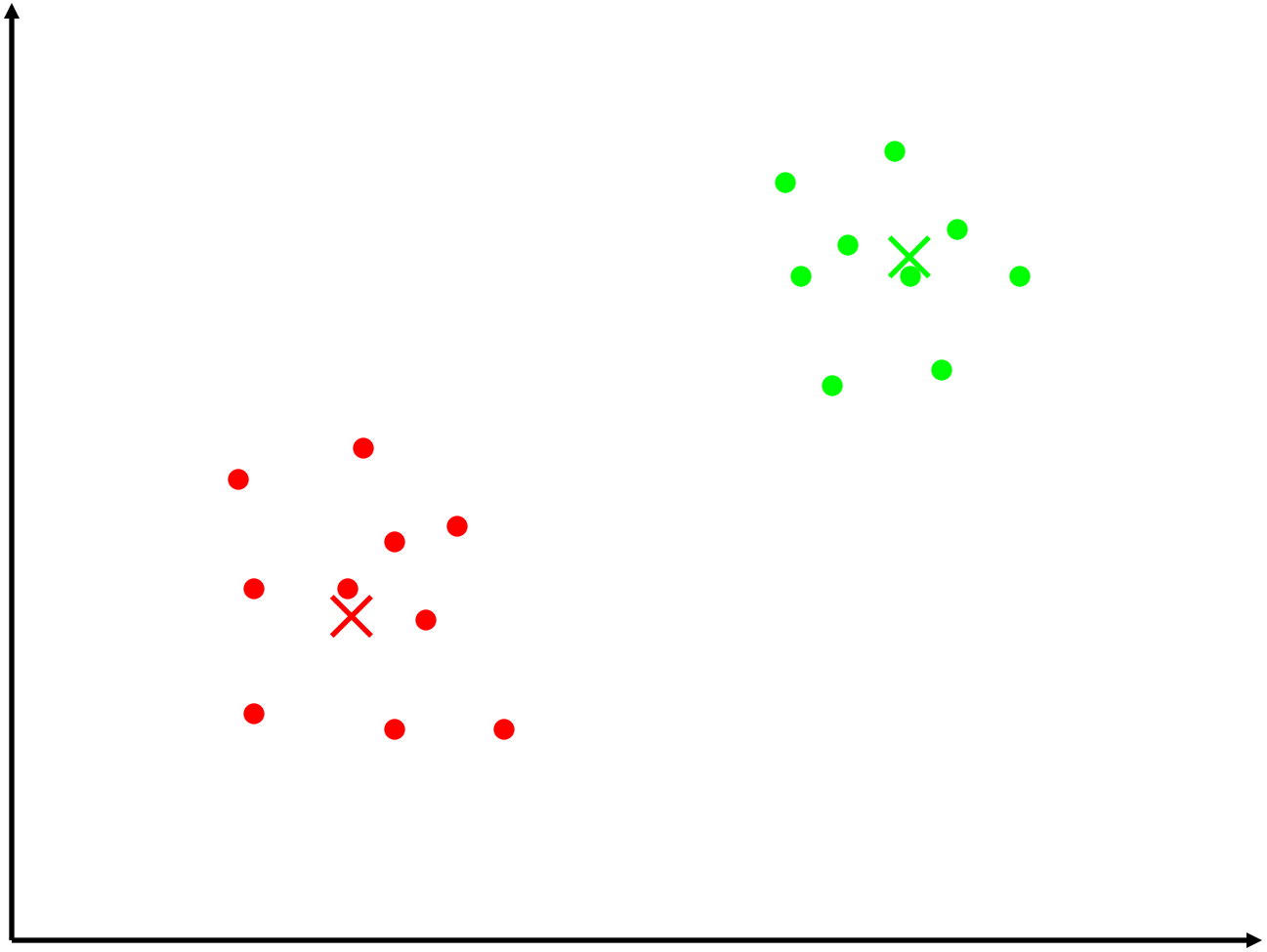
K-Means Clustering



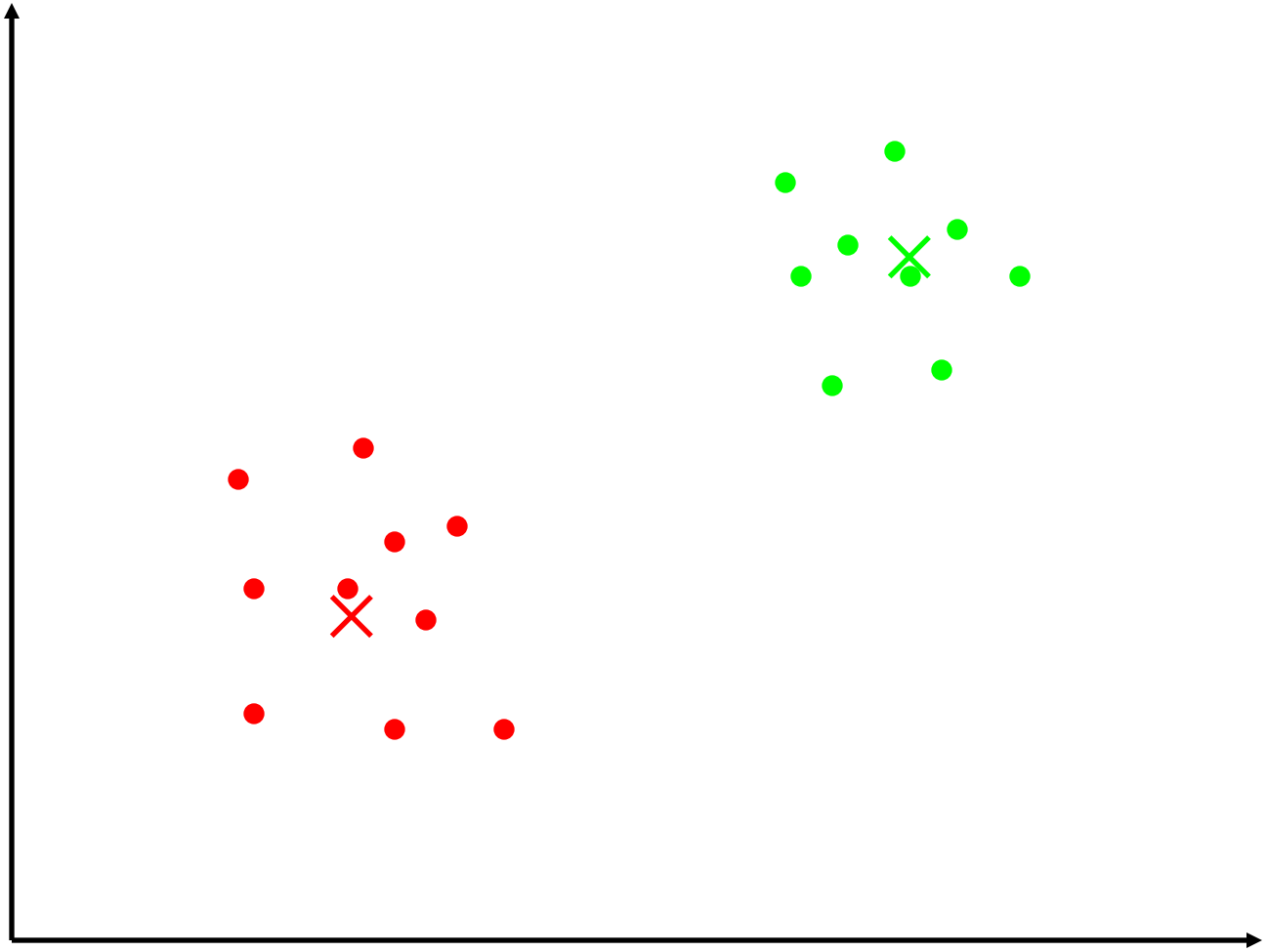
K-Means Clustering



K-Means Clustering

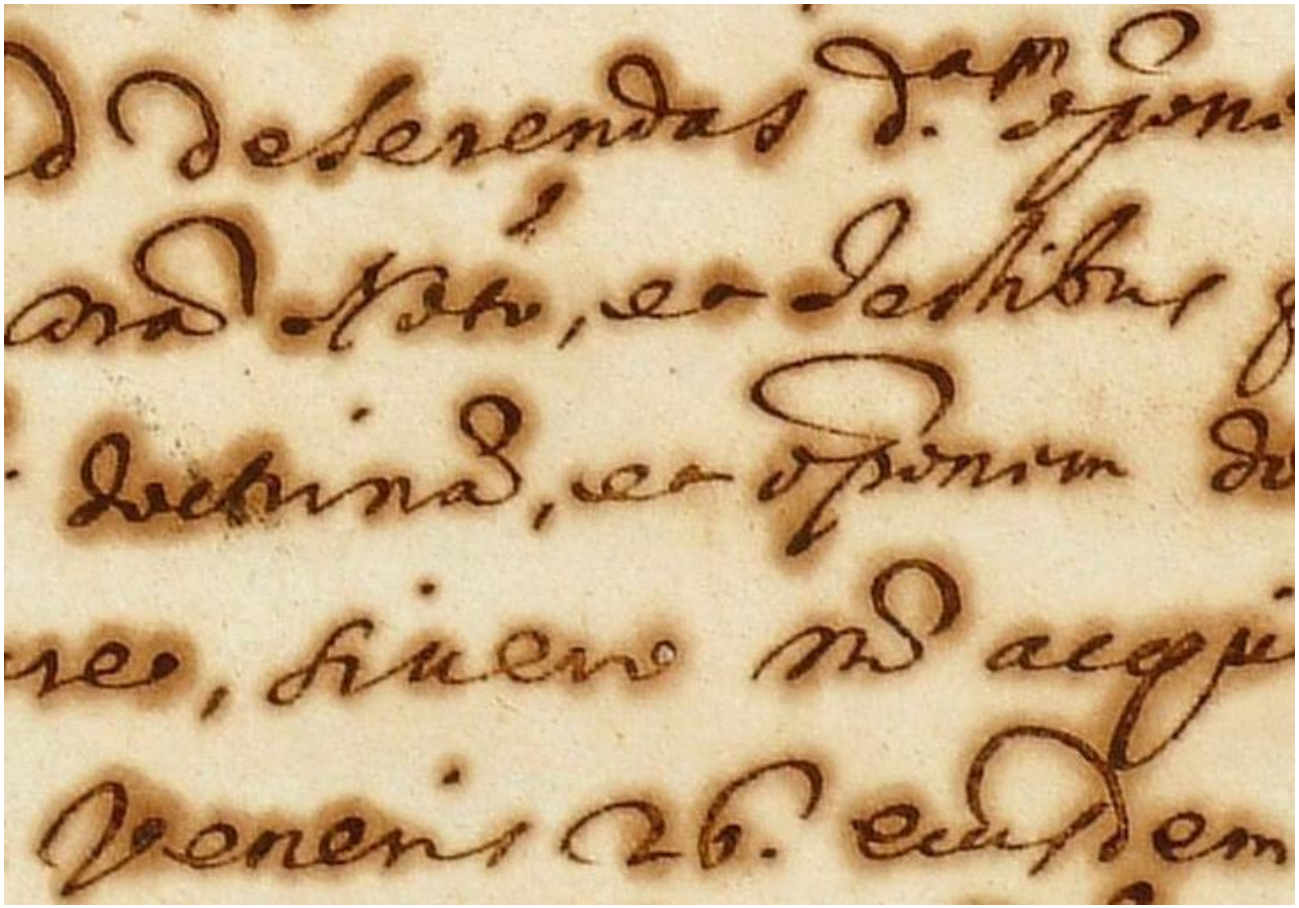


K-Means Clustering



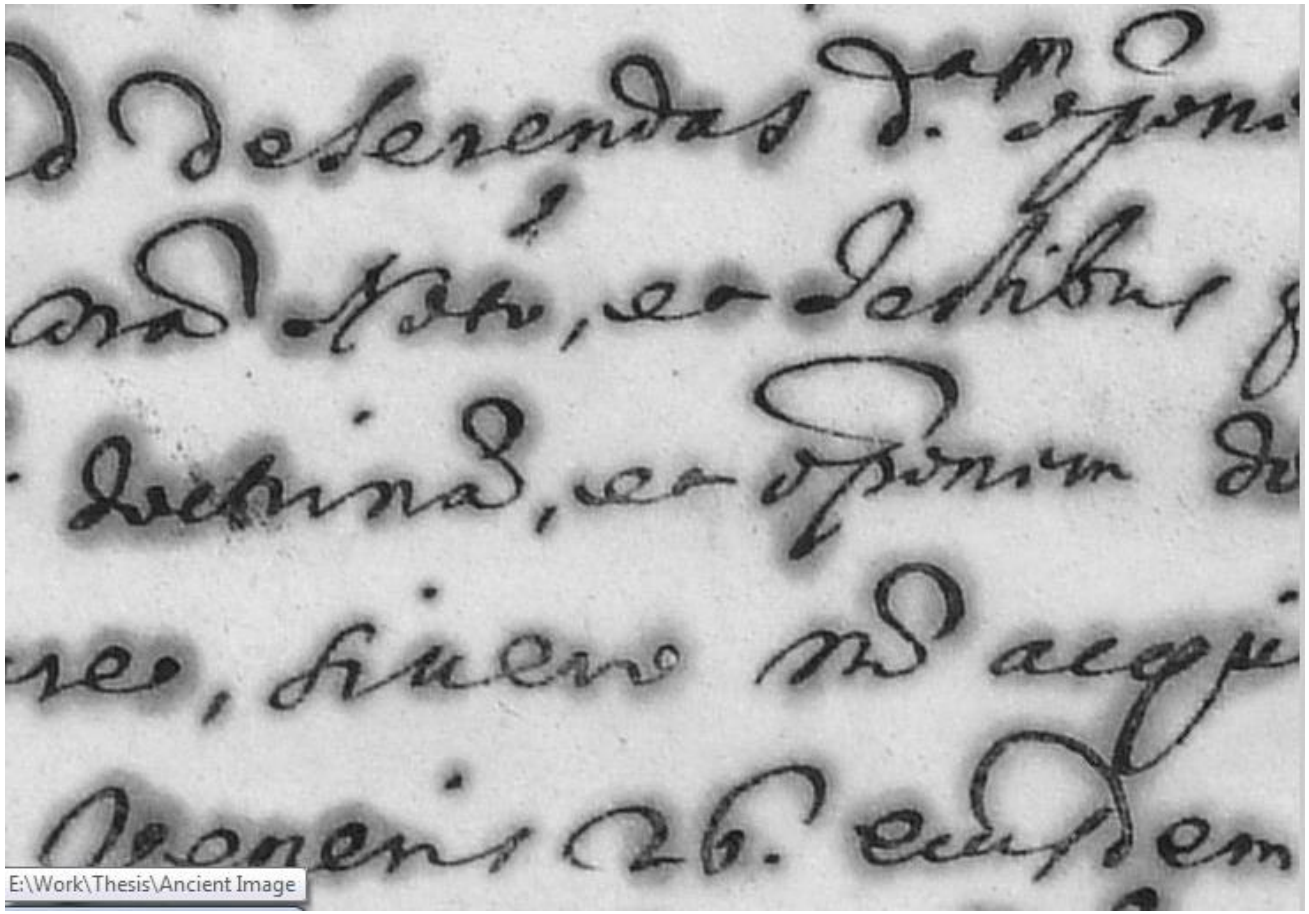
Clustering

- Example



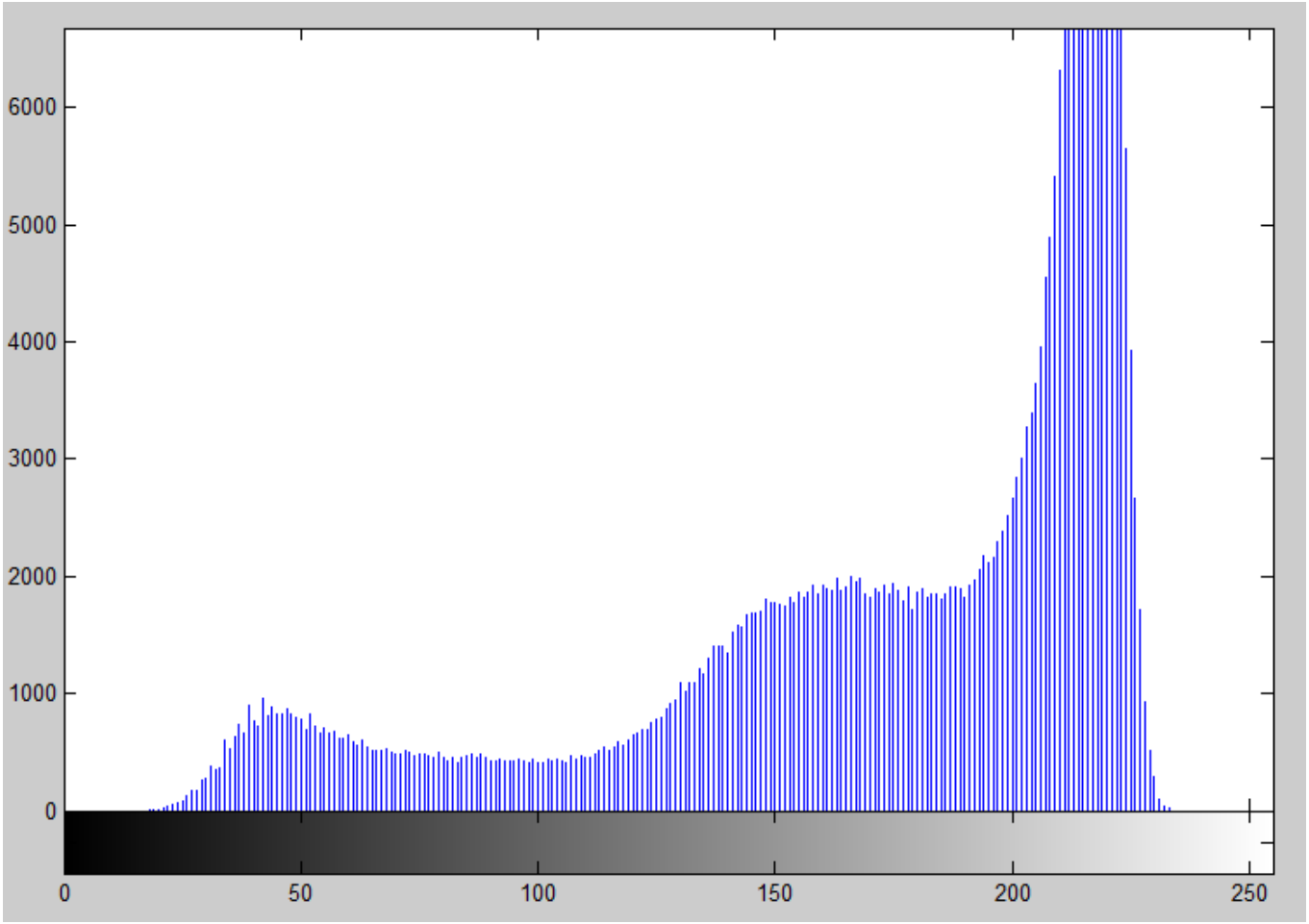
Clustering

- Example



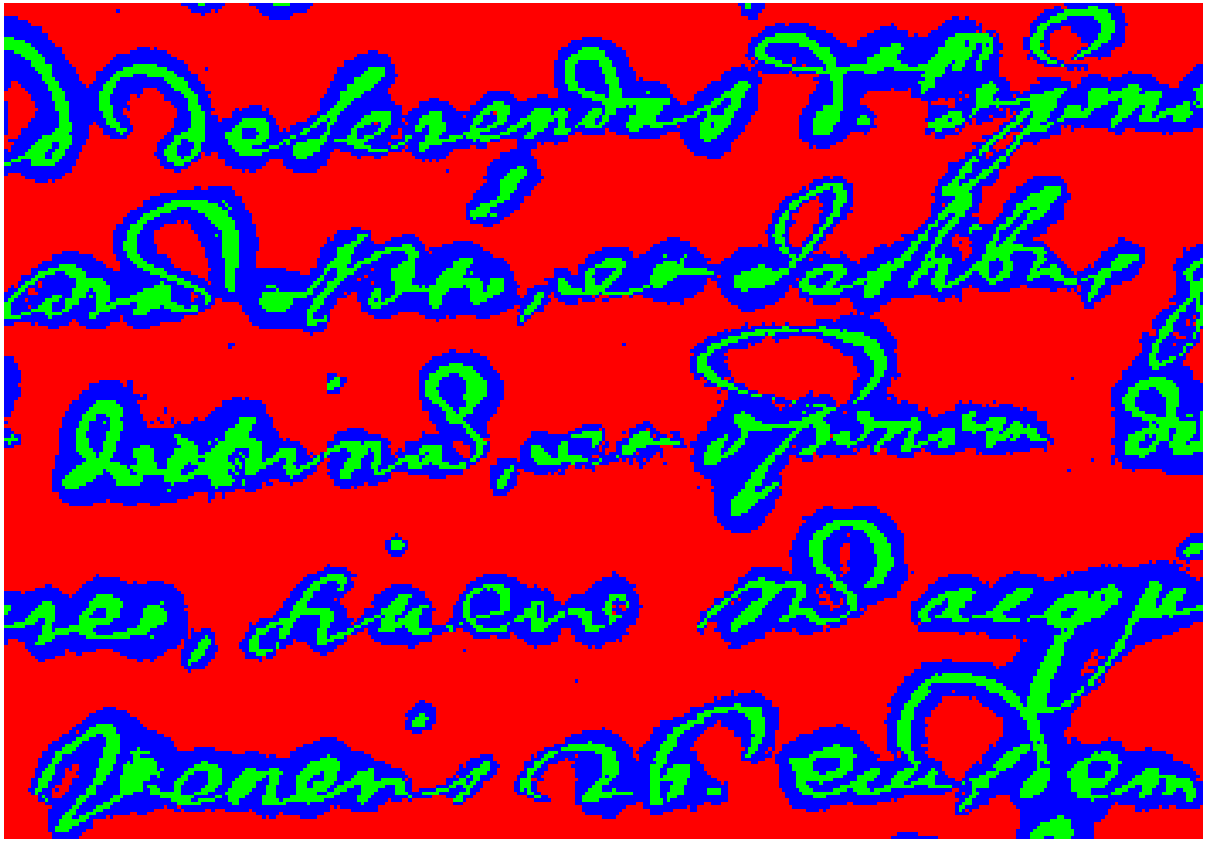
Clustering

- Example



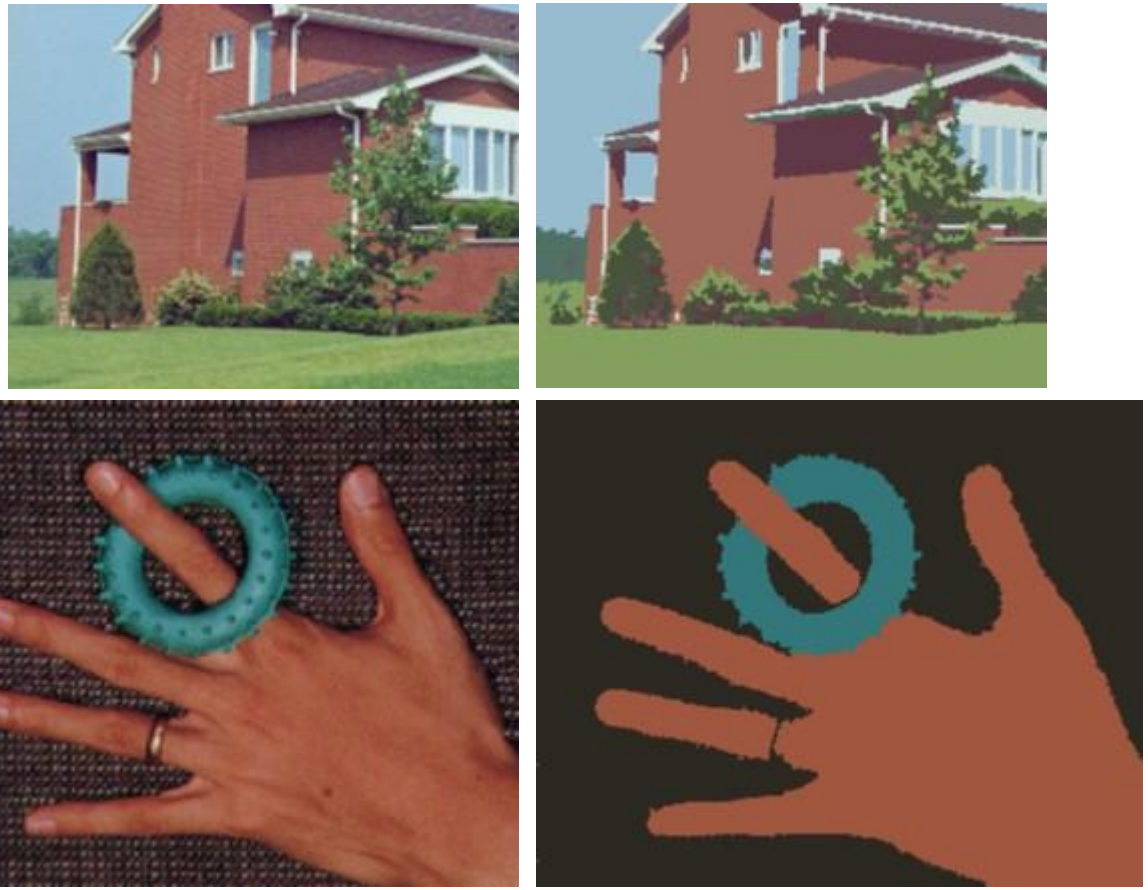
Clustering

- Example



Clustering

- Example



D. Comaniciu and P. Meer, *Robust Analysis of Feature Spaces: Color Image Segmentation*, 1997.

K-Means Clustering

- Example

Original

K=5

K=11

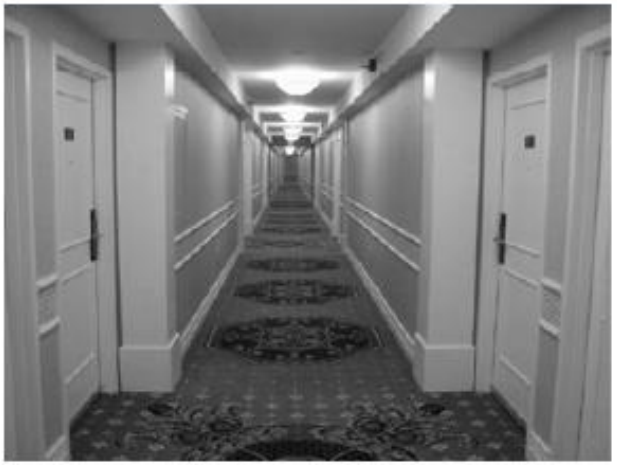
Hough Transform

Introduction to Hough transform

- The Hough transform (HT) can be used to detect lines, circles or other parametric curves.
- It was introduced in 1962 (Hough 1962) and first used to find lines in images a decade later (Duda 1972).
- The goal is to find the location of lines in images.
- This problem could be solved by e.g. Morphology and a linear structuring element, or by correlation.
  - Then we would need to handle rotation, zoom, distortions etc.
- Hough transform can detect lines, circles and other structures if their parametric equation is known.
- It can give robust detection under noise and partial occlusion.

An image with linear structures

- Borders between the regions are straight lines.
- These lines separate regions with different grey levels.
- Edge detection is often used as preprocessing to Hough transform.

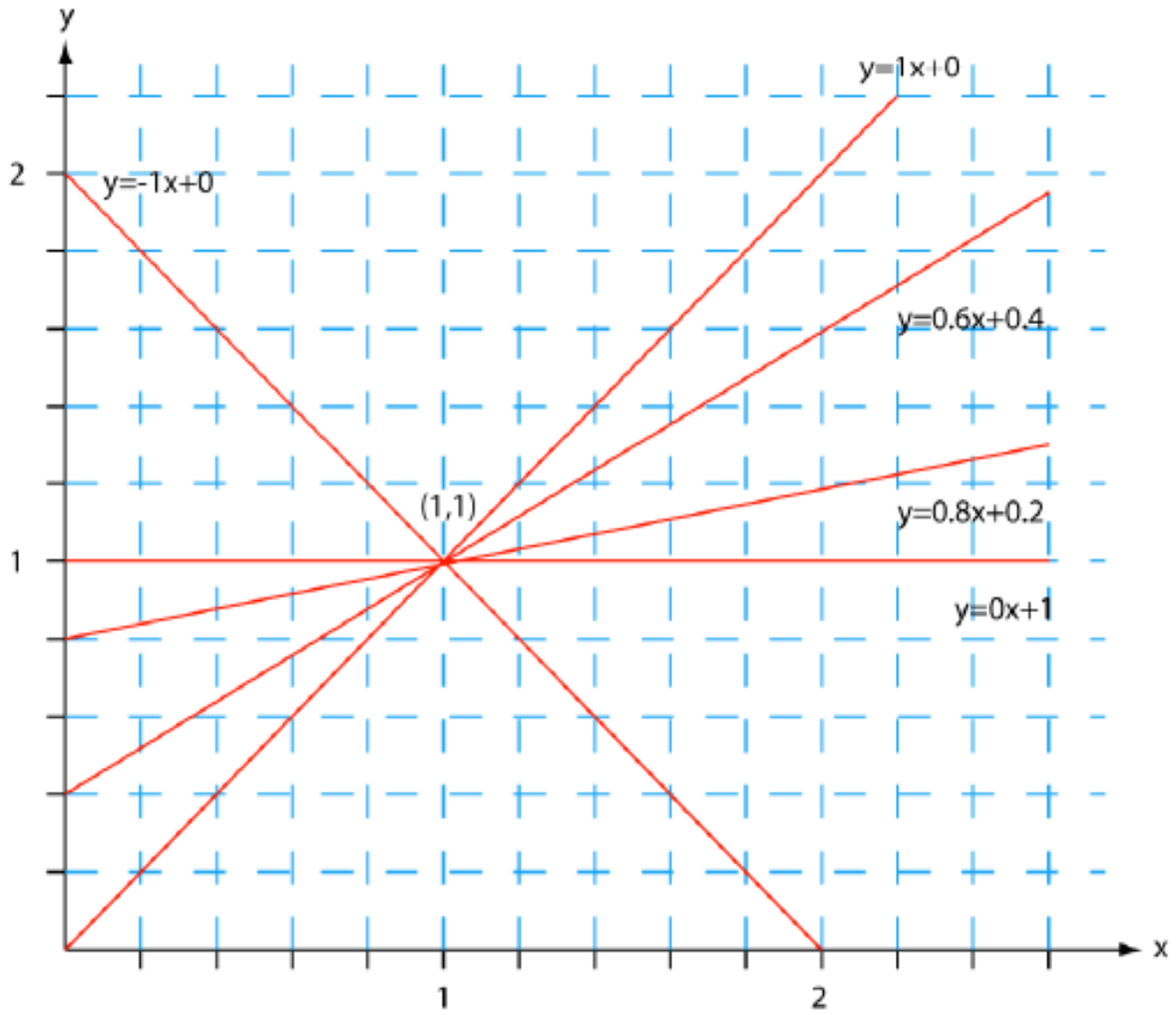


Hough-transform

- Assume that we have performed some edge detection, and a thresholding of the edge magnitude image.
- Thus, we have *n* pixels that may partially describe the boundary of some objects.
- We wish to find sets of pixels that make up straight lines.
- Regard a point (x_i, y_i)

  and a straight line $y_i = ax_i + b$
  - There are many lines passing through the point (x_i, y_i).
  - Common to them is that they satisfy the equation for some set of parameters (a,b).

Hough transform – basic idea



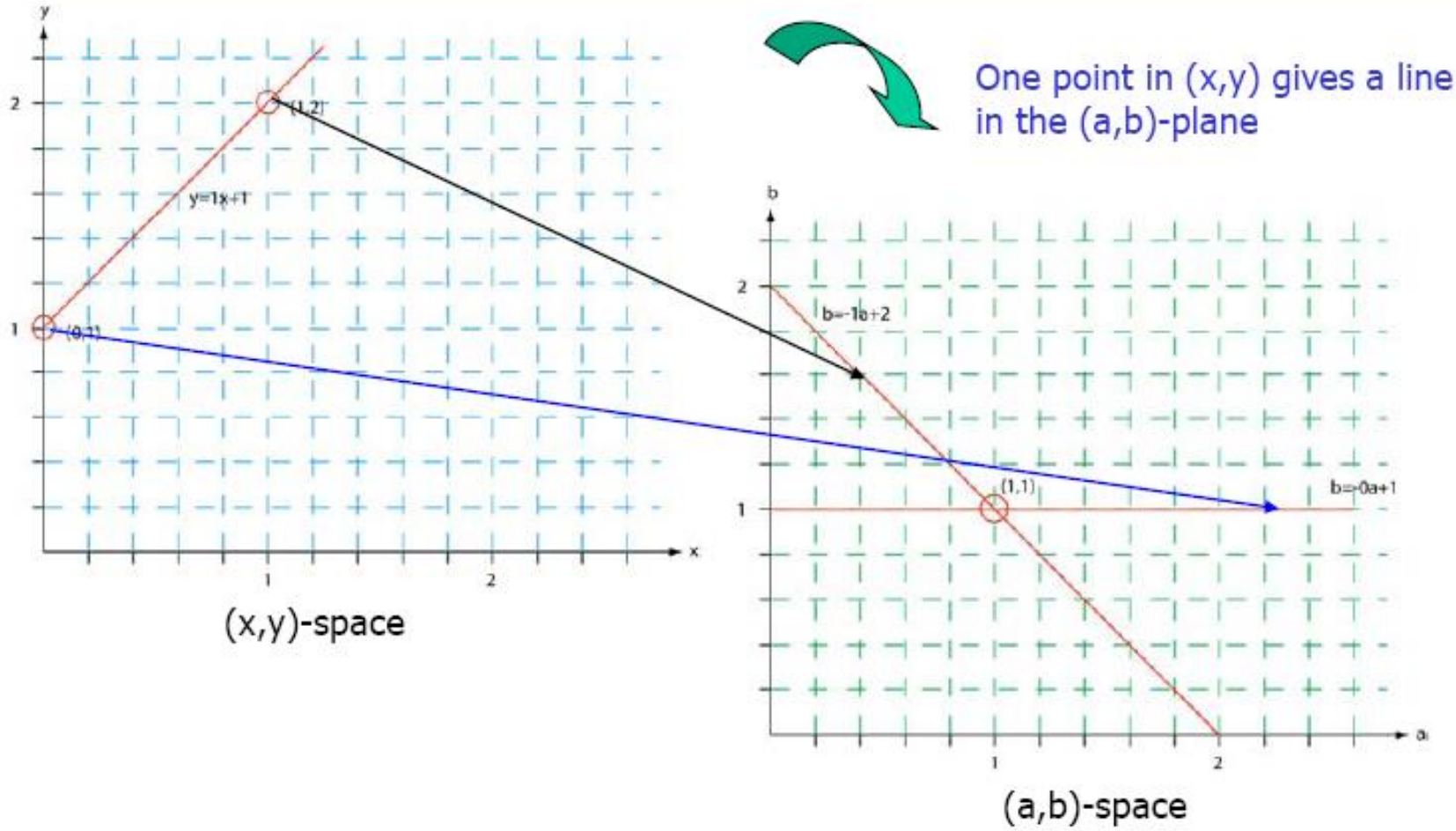
Hough transform – basic idea

- This equation can obviously be rewritten as follows:

$$b = -xa + y$$

- We now consider *x* and *y* as parameters and *a* and *b* as variables.
- This is a line in (*a*,*b*) space parameterized by *x* and *y*.
  - So: a single point in xy-space gives a line in (a,b) space.
- Another point (*x*,*y*) will give rise to another line in (*a*,*b*) space.

Hough transform – basic idea



Hough transform – basic idea

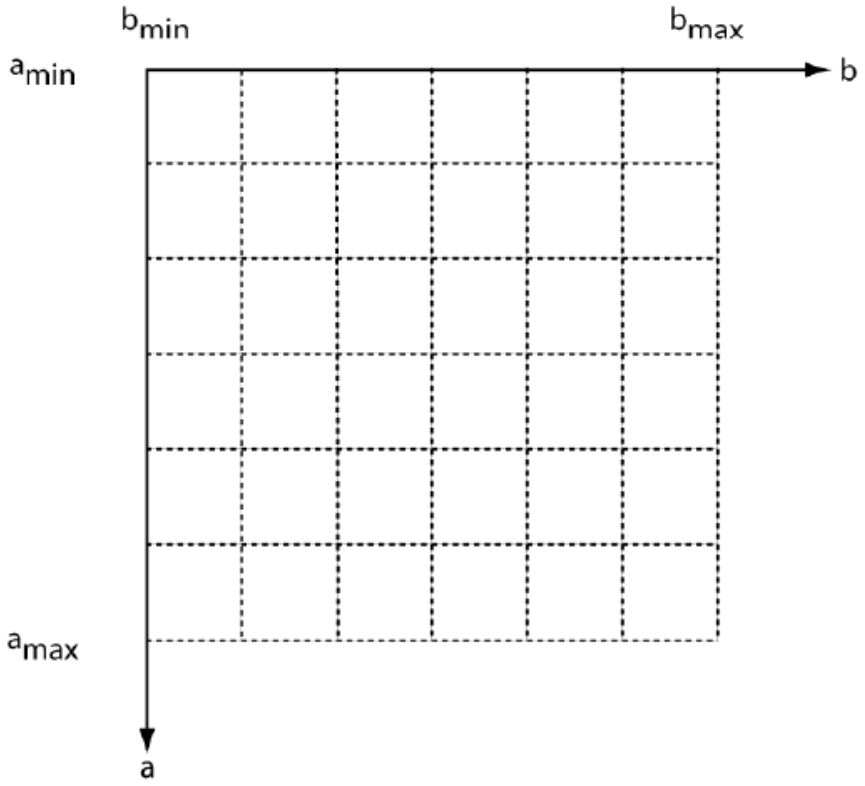
- Two points (x,y) and(z,k) define a line in the (x,y) plane.
- These two points give rise to two different lines in (a,b) space.
- In (a,b) space these lines will intersect in a point (a′,b′) where a′ is the rise and b′ the intersect of the line defined by (x,y) and (z,k) in (x,y) space.
- The fact is that all points on the line defined by (x,y) and (z,k) in (x,y) space will parameterize lines that intersect in (a′,b′) in (a,b) space.
- Points that lie on a line will form a "cluster of crossings" in the (a,b) space.

Hough transform – algorithm

- Quantize the parameter space (a,b), that is, divide it into cells.
- This quantized space is often referred to as the accumulator cells.
- In the figure in the next slide a_{min} is the minimal value of a etc.
- Count the number of times a line intersects a given cell.
  - For each point (x,y) with value 1 in the binary image, find the values of (a,b) in the range $[[a_{min},a_{max}],[b_{min},b_{max}]]$ defining the line corresponding to this point.
  - Increase the value of the accumulator for these [a',b'] point.
  - Then proceed with the next point in the image.

- Cells receiving a minimum number of "votes" are assumed to correspond to lines in (x,y) space.
  - Lines can be found as peaks in this accumulator space.

Hough transform - algorithm



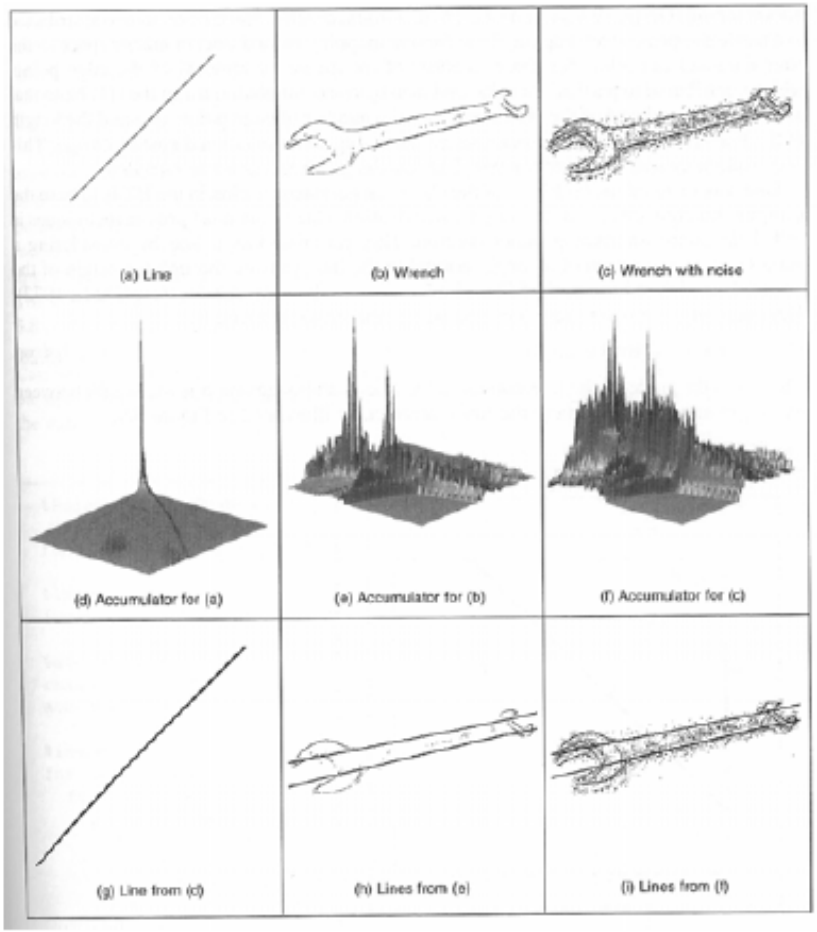
Hough accumulator cells

Example – images and accumulator space

Thresholded edge images

Visualizing the accumulator space
The height of the peak will be defined by the number of pixels in the line.

Thresholding the accumulator space and superimposing this onto the edge image

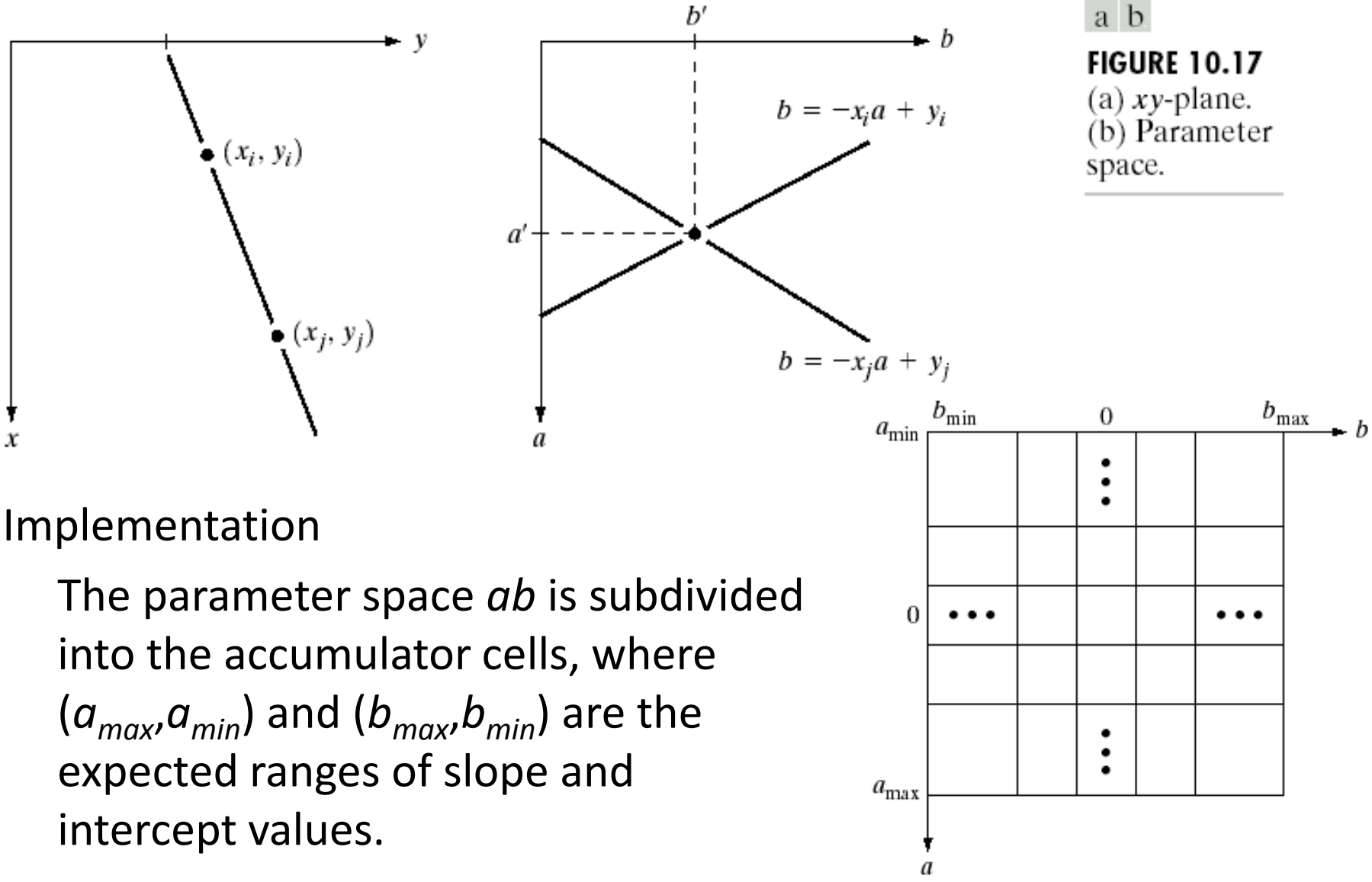


Note how noise effects the accumulator. Still with noise, the largest peaks correspond to the major lines.

Hough Transform

- The edge points are linked by determining if they lie on a curve of specified shape.
- Let us take the case of straight lines: we want to link points if they lie on a straight line.
- Consider a point (x_i,y_i), the equation of any line passing through this point is given by: $y_i = a\,x_i + b$, which can be written as: $b = -\,x_i a + y_i$. Therefore every point in xy plane corresponds to a straight line in ab plane.
- If there is a second point (x_j,y_j), another line with equation: $b = -\,x_j a + y_j$ is drawn in the ab plane.
- The intersection of the two lines in ab space give the values of (a',b'), which define a line passing through both points (x_i,y_i) and (x_j,y_j).

Hough Transform



a b

FIGURE 10.17
(a) *xy*-plane.
(b) Parameter space.

Implementation

The parameter space *ab* is subdivided into the accumulator cells, where (a_{max},a_{min}) and (b_{max},b_{min}) are the expected ranges of slope and intercept values.

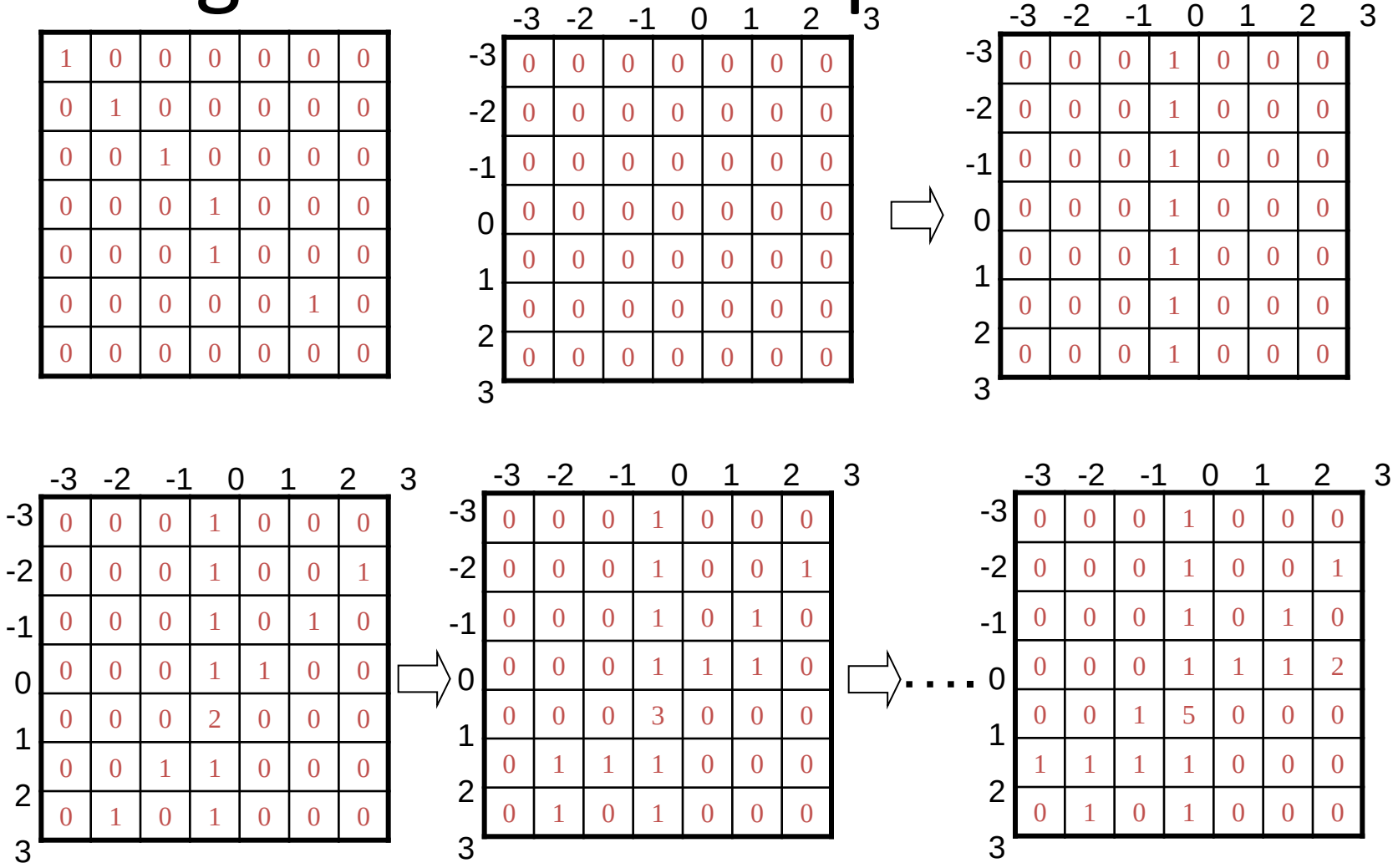
Hough Transform

Implementation steps:

- Initially the accumulator cells are set to zero.
- Then for every point (x_k,y_k) in the image, a is varied over the allowed subdivision values and the corresponding b values are calculated using $b = -x_k a + y_k$.
- The resulting b are then rounded off to the allowed values of b.
- If a value of a_p results in solution b_q, we let the corresponding accumulator value $A(p,q) = A(p,q) +1$.
- In the end the value of Q in $A(i,j)$ corresponds to Q points on the line $y = -a_i x + b_j$.

  Note: The number of subdivisions in the ab plane determines the accuracy of the colinearity of these points.

Hough Transform Implementation

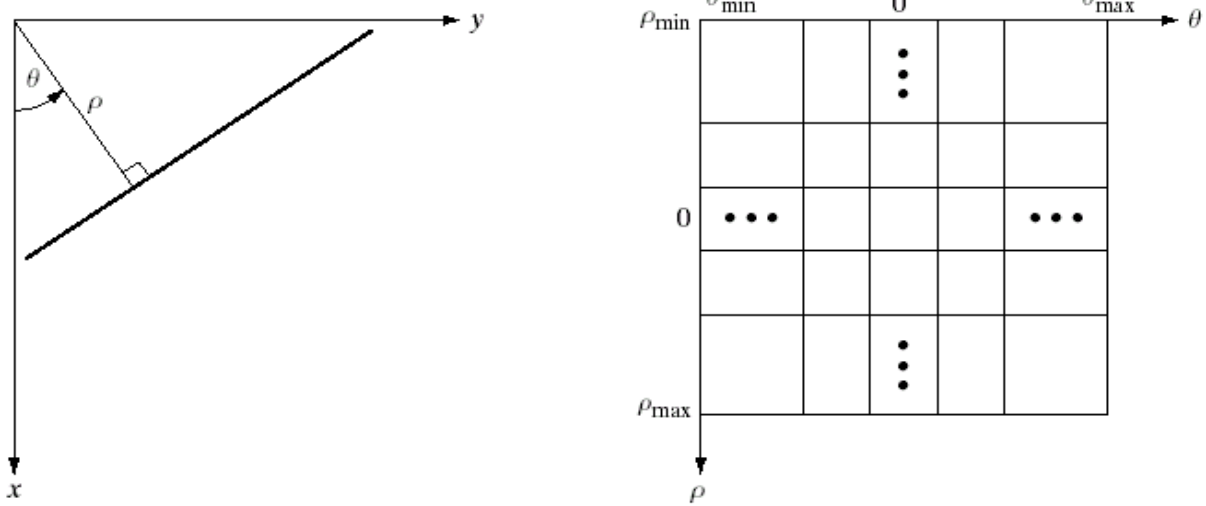


Parameters a and b can take values between –infinity to +infinity

Hough Transform

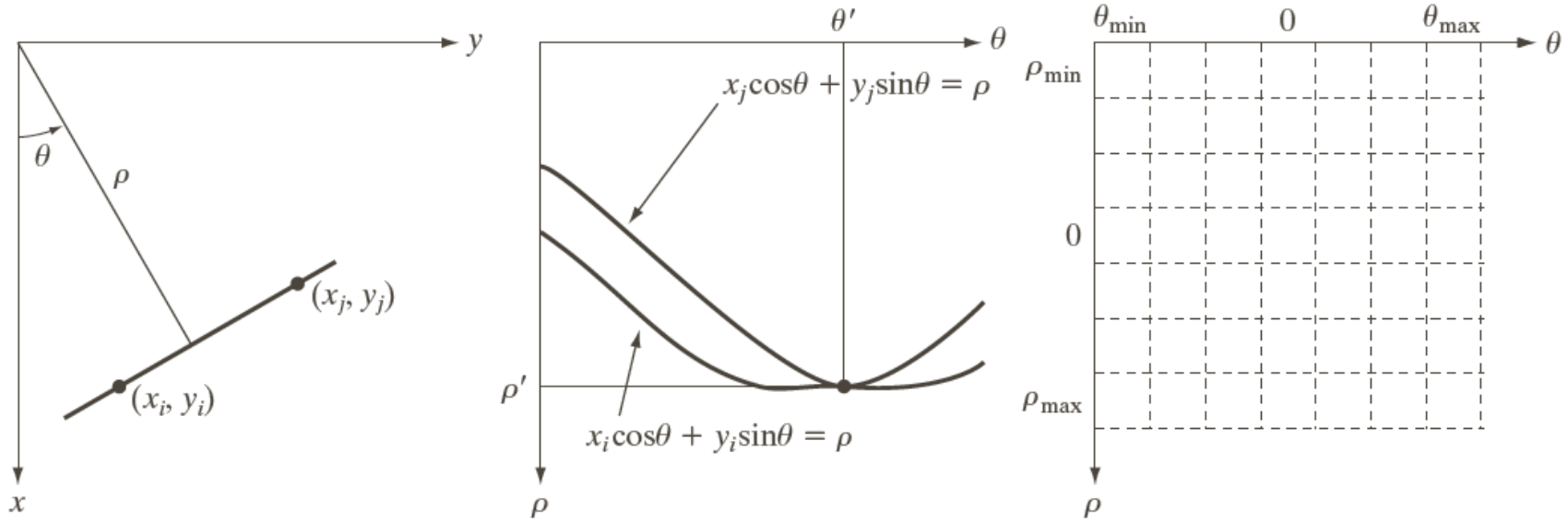
- Limitation in using $y_i = a\,x_i + b$, as the representation of straight line is that slope approaches to infinity as the line approaches to be vertical.
- Therefore usually Hough Transform is implemented using the polar equation of straight line, i.e. $x\cos\theta + y\sin\theta = \rho$

  and instead of ab-plane $\rho\theta$-plane is used.
- Every point in image gives a sinusoidal curve in the $\rho\theta$-plane.



a b

FIGURE 10.19
(a) Normal representation of a line.
(b) Subdivision of the $\rho\theta$-plane into cells.



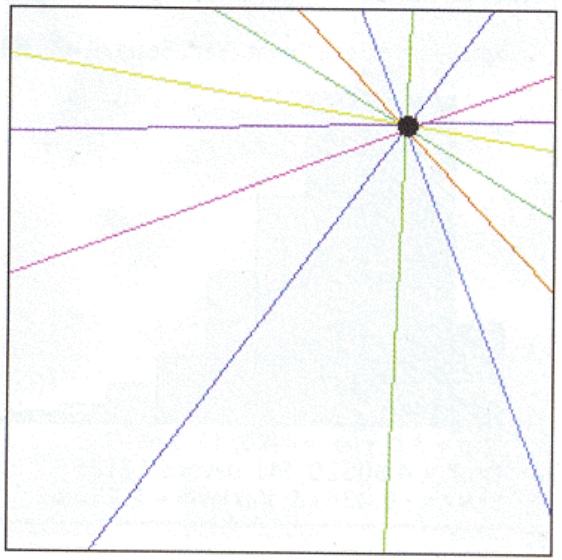
a b c

FIGURE 10.32 (a) (ρ, θ) parameterization of line in the xy-plane. (b) Sinusoidal curves in the $\rho\theta$-plane; the point of intersection (ρ', θ') corresponds to the line passing through points (x_i, y_i) and (x_j, y_j) in the xy-plane. (c) Division of the $\rho\theta$-plane into accumulator cells.

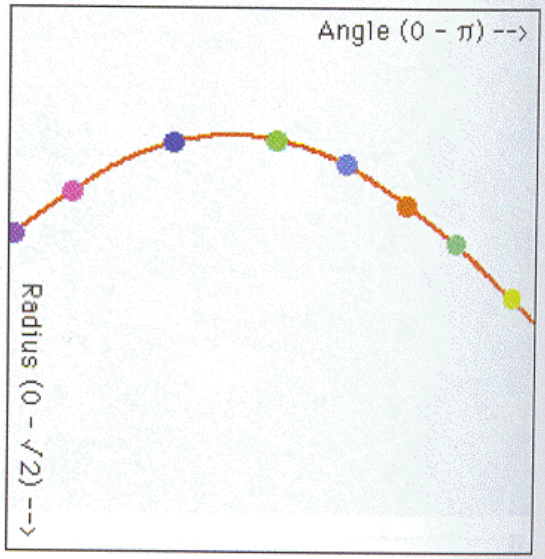
See Example on You Tube:
https://www.youtube.com/watch?v=4zHbI-fFIlI

Hough Transform

Principle of the Hough transform. *Each point in the real-space image* ***(a)*** *produces a sinusoidal line in Hough space* ***(b)*** *representing all possible lines that can be drawn through it. Each point in Hough space corresponds to a line in real space. The real-space lines corresponding to a few of the points along the sinusoid are shown, with color coding to match them to the points.*



a



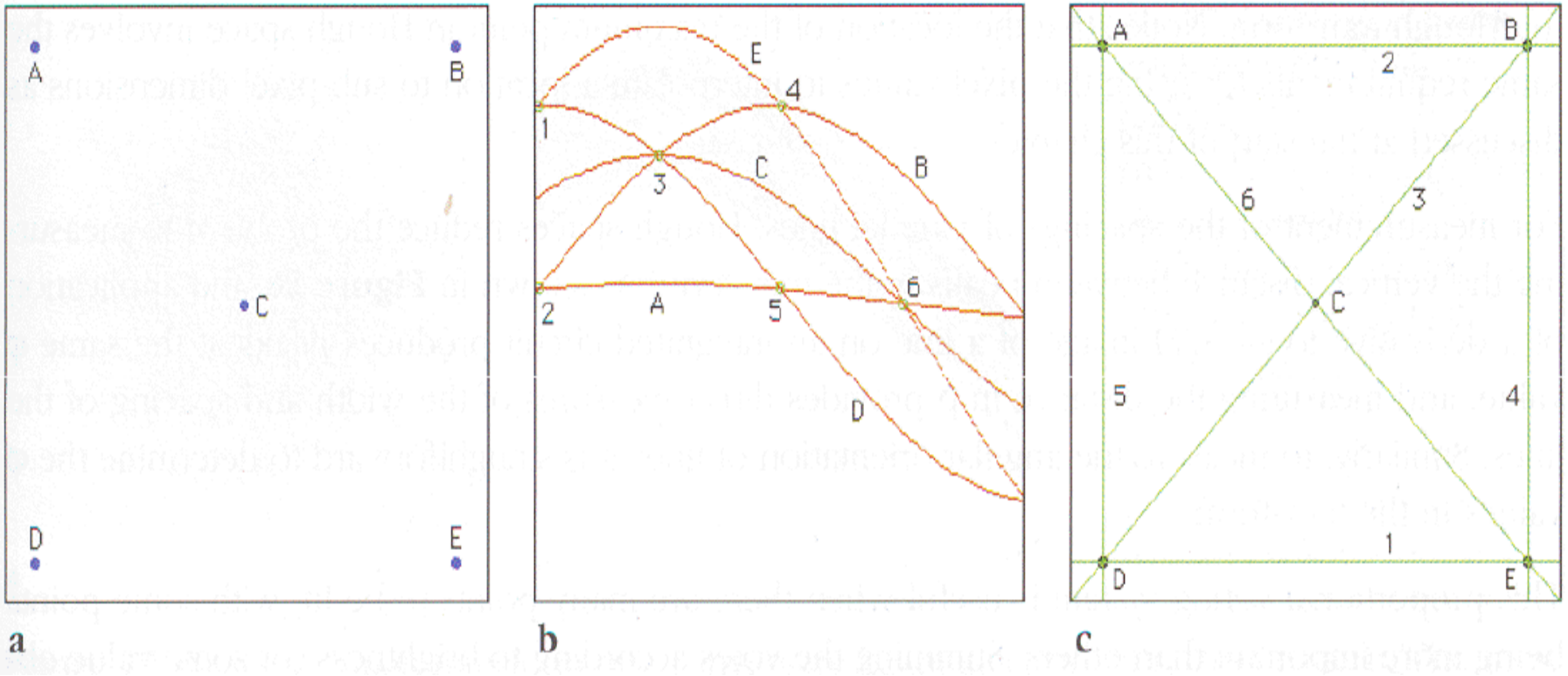
b

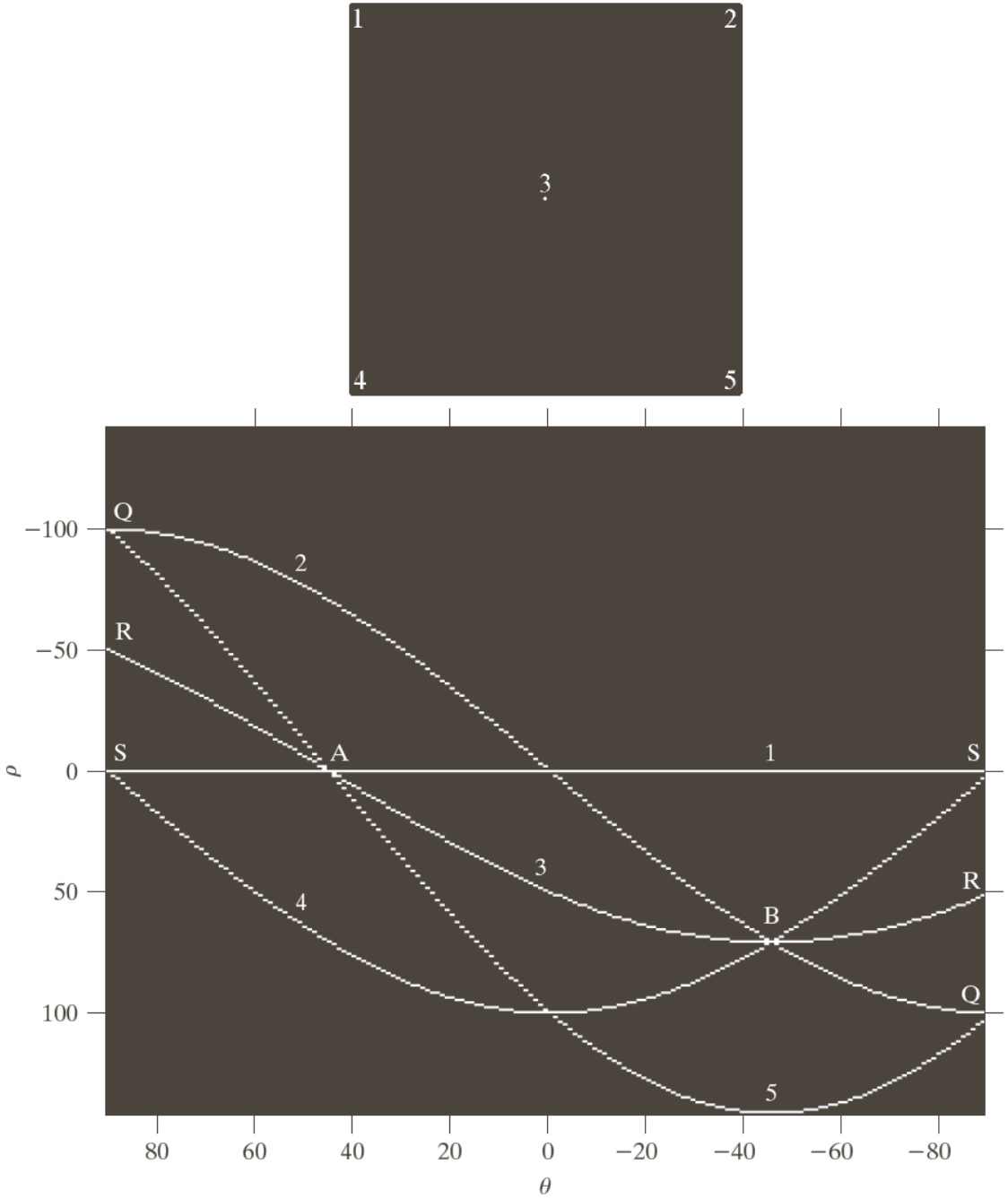
Hough Transform





Example

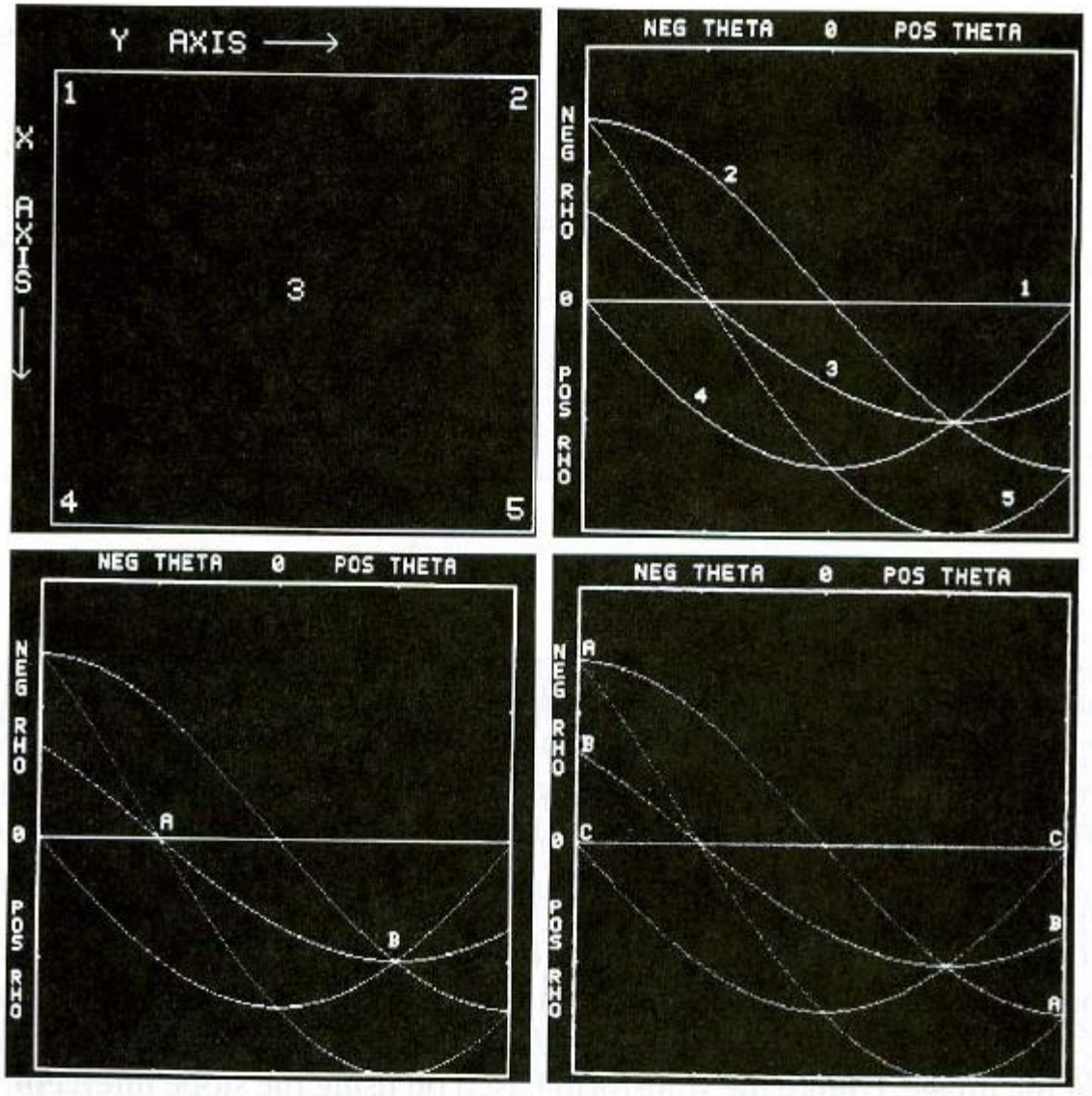




a
b

FIGURE 10.33
(a) Image of size 101×101 pixels, containing five points.
(b) Corresponding parameter space. (The points in (a) were enlarged to make them easier to see.)

Hough Transform



Implementation of the Hough transform

- Construct an array representing θ, ρ
- For each point, render the curve (θ, ρ) into this array, adding one at each cell
- Difficulties
  - how big should the cells be? (too big, and we cannot distinguish between quite different lines; too small, and noise causes lines to be missed)
- How many lines?
  - count the peaks in the Hough array
- This method can be extended to other type of curves also by using the equation for the desired curve, e.g. circle:

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2$$

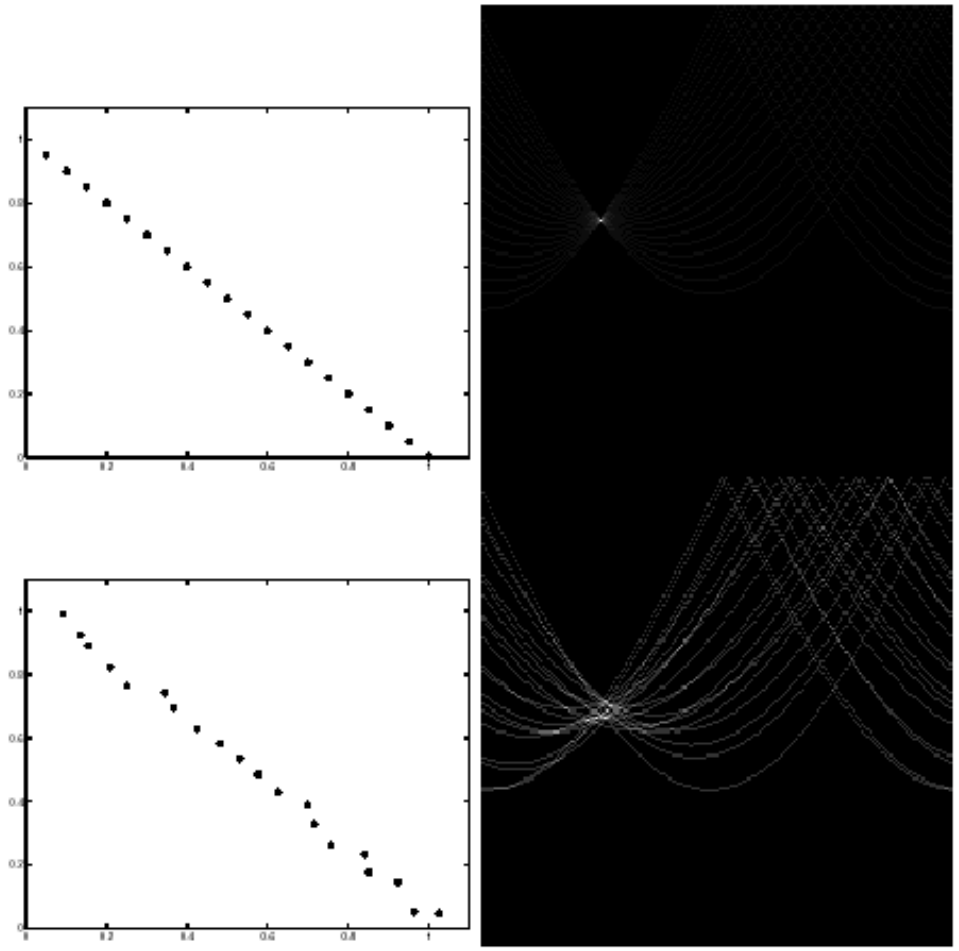


Figure 16.2. The Hough transform maps each point like token to a curve of possible lines (or other parametric curves) through that point. These figures illustrate the Hough transform for lines. The left hand column shows points, and the right hand column shows the corresponding accumulator arrays (the number of votes is indicated by the grey level, with a large number of votes being indicated by bright points). The top shows a set of 20 points drawn from a line next to the accumulator array for the Hough transform of these points. Corresponding to each point is a curve of votes in the accumulator array; the largest set of votes is 20. The horizontal variable in the accumulator array is θ and the vertical variable is r; there are 200 steps in each direction, and r lies in the range $[0, 1.55]$. In the center, these points have been offset by a random vector each element of which is uniform in the range $[0, 0.05]$; note that this offsets the curves in the accumulator array shown next to the points; the maximum vote is now 6.

Hough Transform

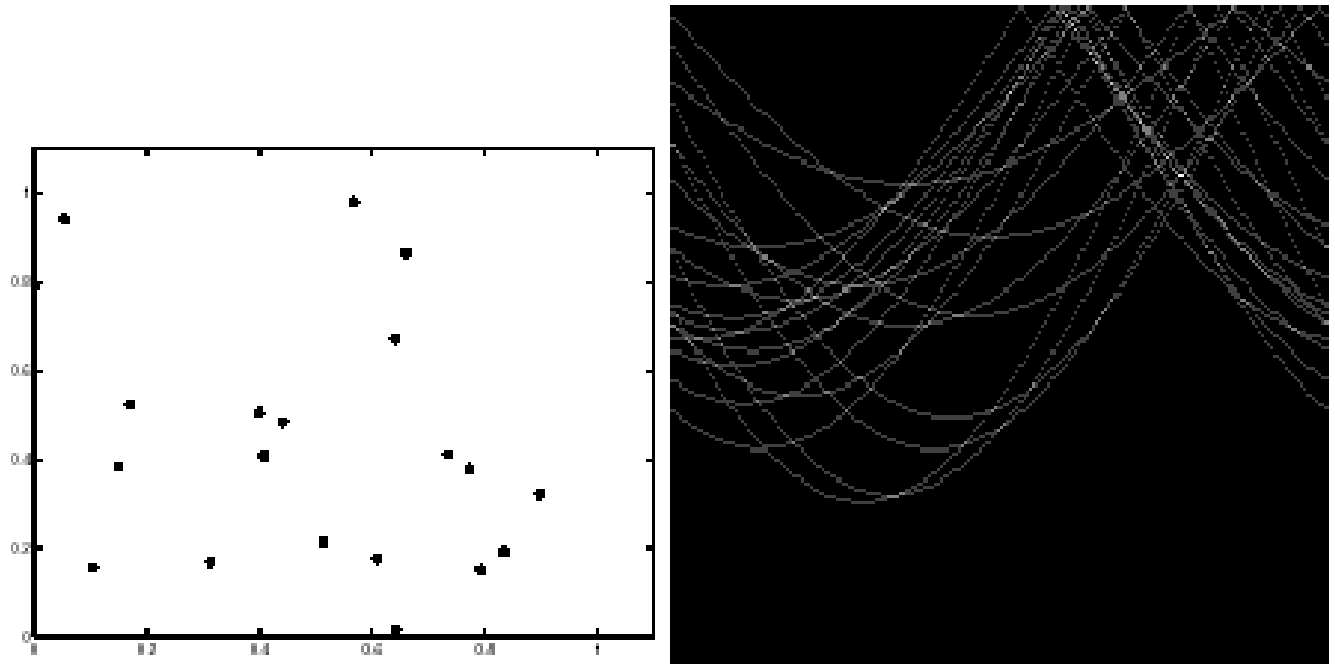
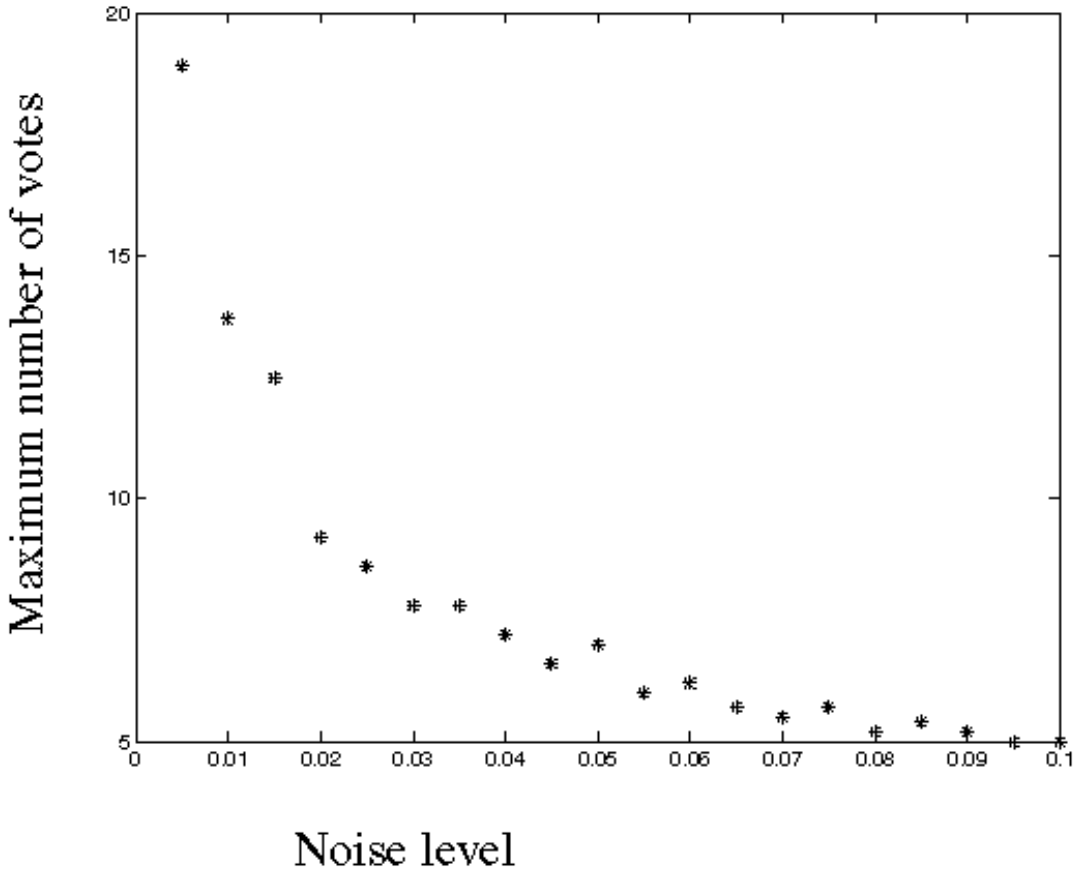


Figure 16.3. The Hough transform for a set of random points can lead to quite large sets of votes in the accumulator array. As in figure 16.2, the left hand column shows points, and the right hand column shows the corresponding accumulator arrays (the number of votes is indicated by the grey level, with a large number of votes being indicated by bright points). In this case, the data points are noise points (both coordinates are uniform random numbers in the range $[0,1]$); the accumulator array in this case contains many points of overlap, and the maximum vote is now 4. Figures 16.4 and explore noise issues somewhat further.

Noise Limitations of Hough Transform

- Two main limitations:noise and cell size



Results for a specimen of line with 20 points, with different amounts of noise

Acknowledgements

- Digital Image Processing", Rafael C. Gonzalez & Richard E. Woods, Addison-Wesley, 2002
- Computer Vision for Computer Graphics, Mark Borg

Material in these slides has been taken from, the following resources