OpenCV

Open Source Computer Vision Library

- detect and recognize faces
- identify objects
- classify human actions
- track camera movements
- track moving objects
- extract 3D models of objects
- produce 3D point clouds from stereo cameras
- stitch images together
- find similar images from an image database
- deep learning
- image classification
- …

For 7785…

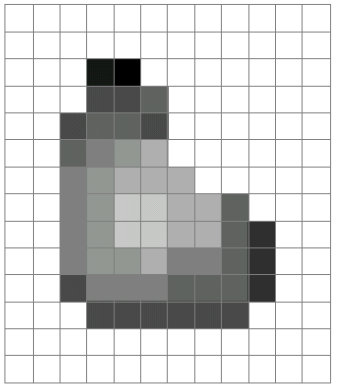
- We will use version 3.3
- OpenCV is available in both C++ and Python

Image Processing Crash Course

(particularly circle and line fitting)

Images

- A grid (matrix) of intensity values



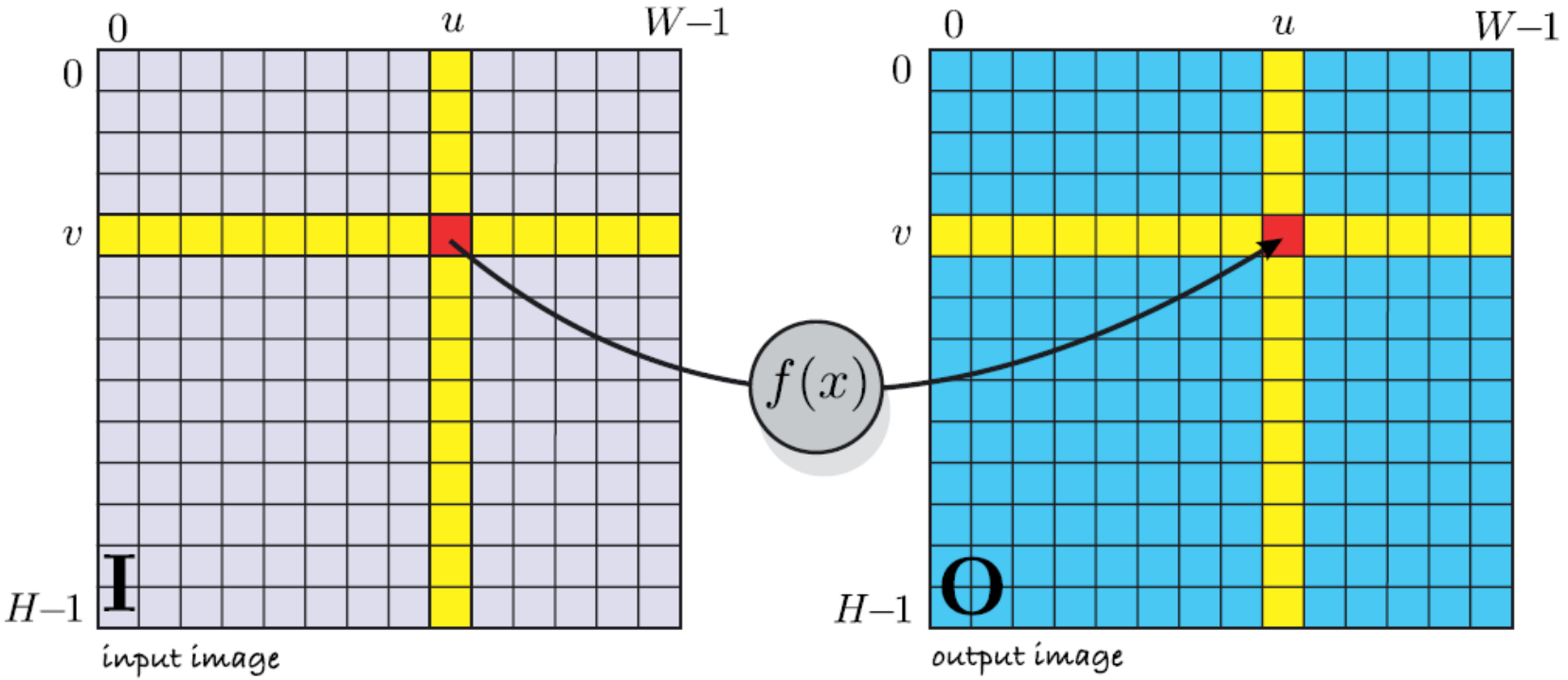
=

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

- (common to use one byte per value: 0 = black, 255 = white)

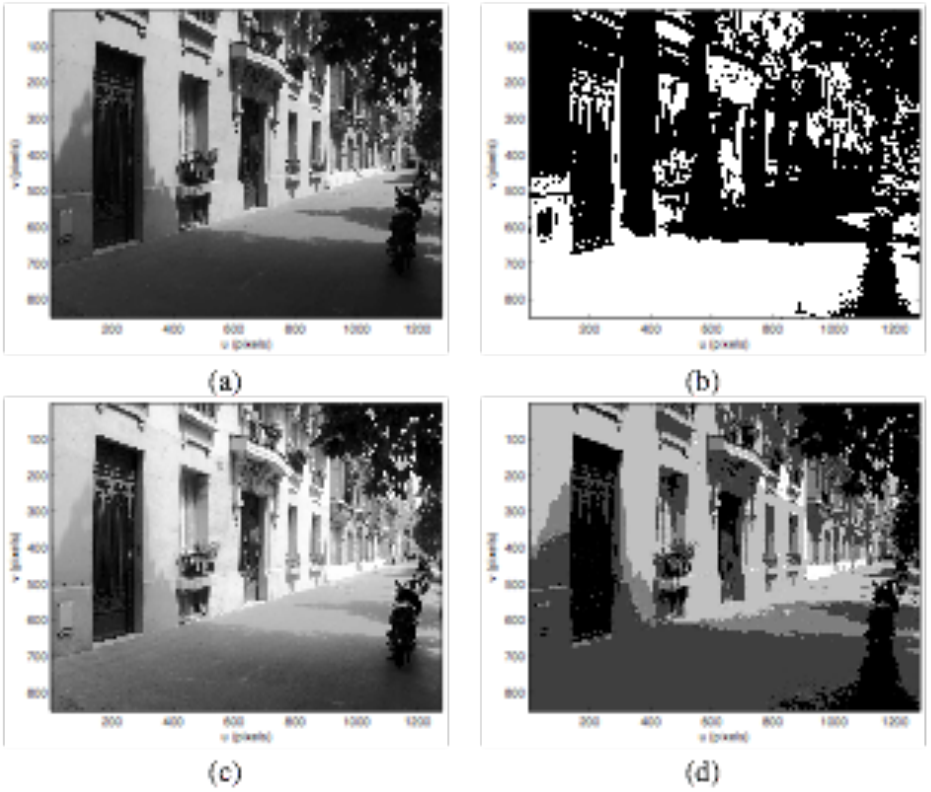
Monadic operators

Monadic image processing operations. Each output pixel is a function of the corresponding input pixel (shown in red)



1-1 mapping between pixels

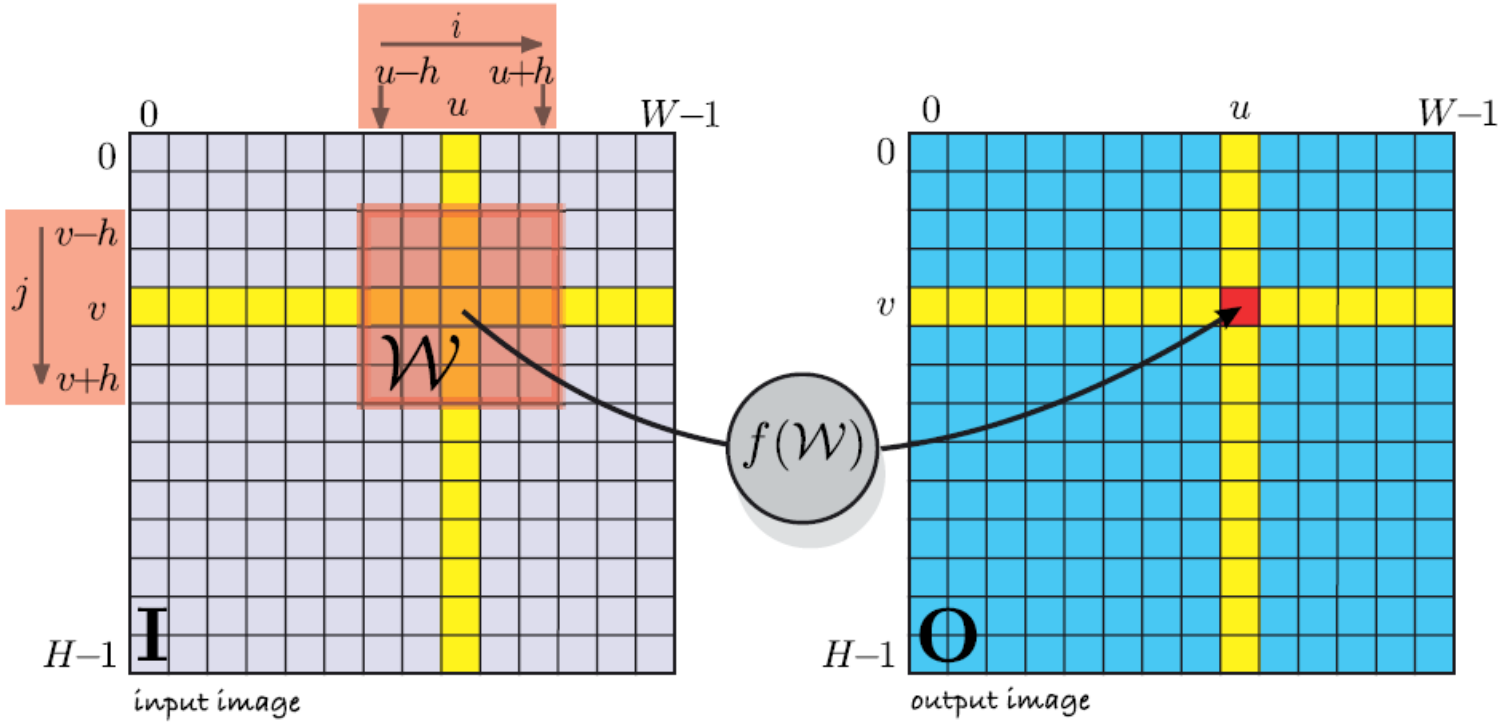
Signal enhancement



Some monadic image operations, **a** original, **b** shadow regions, **c** histogram normalized, **d** posterization

Local operators

Local image processing operations. The reddish shaded region on the left shows the window W that is the set of pixels used to compute the red output pixel on the right



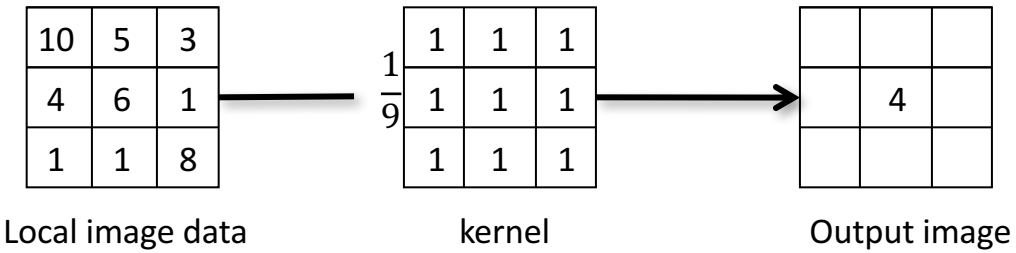
many-to-one mapping between pixels

Linear filtering

- Replace each pixel by a linear combination of its neighbors
- The matrix of the linear combination is called the “kernel,” “mask”, or “filter”

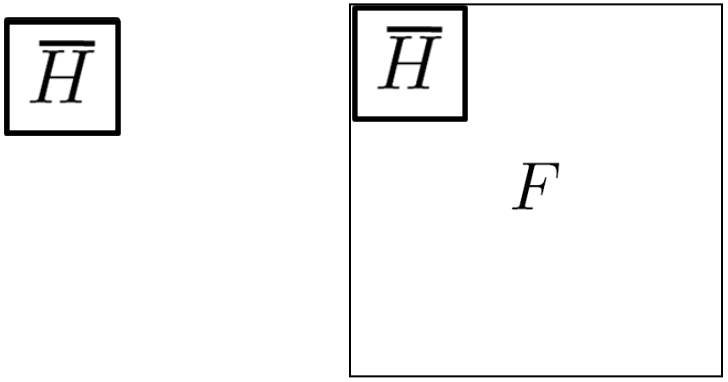
Let F be the image, H be the kernel (of size $2k+1$ by $2k+1$), and G be the output image

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$



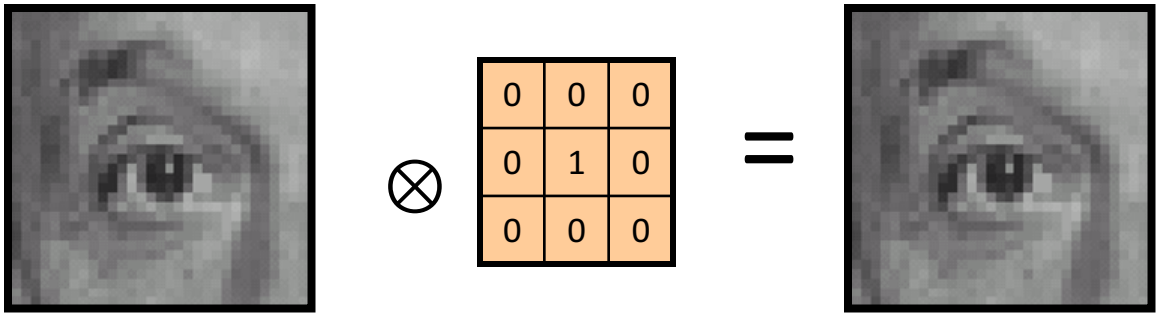
Convolution

- Applying a kernel to an image in this way is called convolution



Adapted from F. Durand

Linear filters: examples

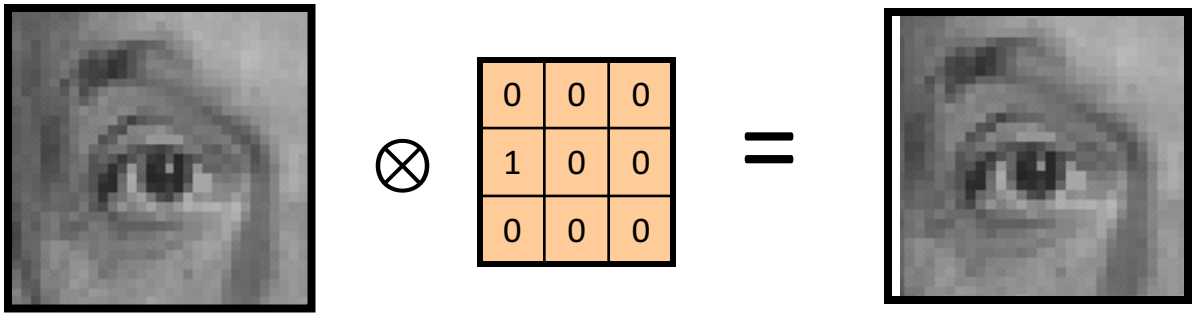


Original

Identical image

Source: D. Lowe

Linear filters: examples

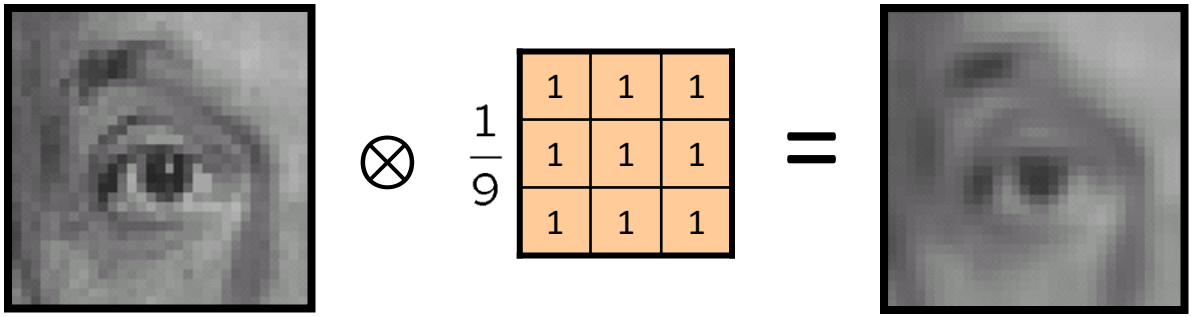


Original

Shifted right
By 1 pixel

Source: D. Lowe

Linear filters: examples

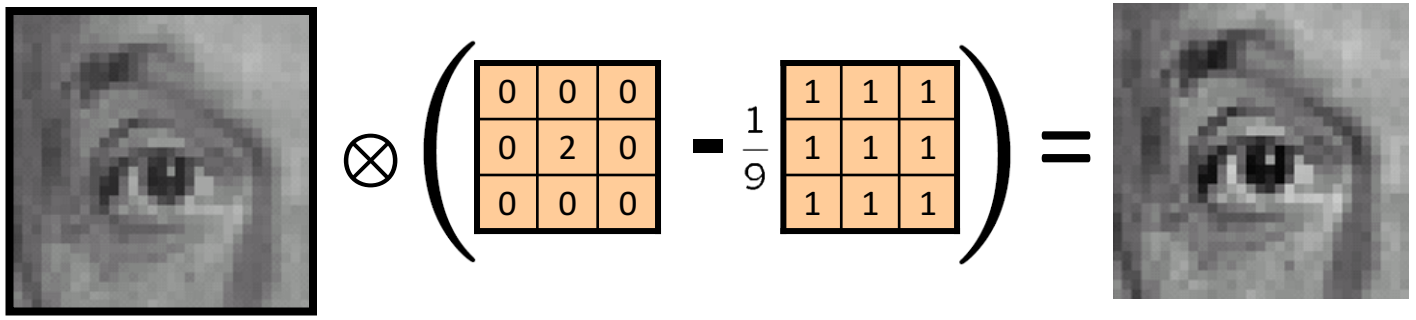


Original

Mean filter (blurring)

Source: D. Lowe

Linear filters: examples

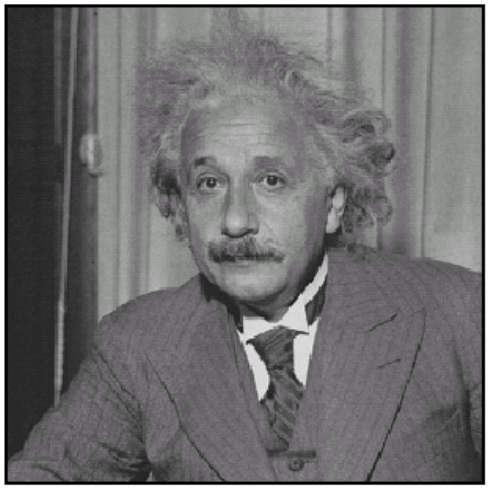


Original

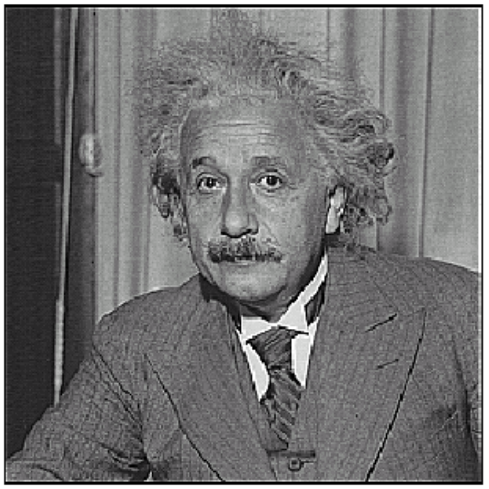
Sharpening filter
(accentuates edges)

Source: D. Lowe

Sharpening



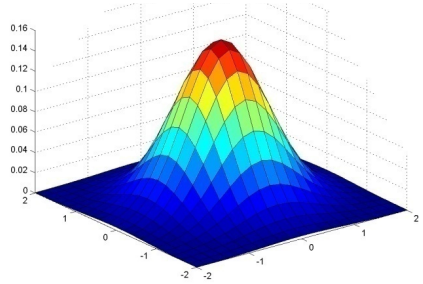
before

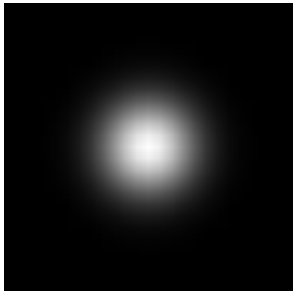


after

Source: D. Lowe

Gaussian Kernel





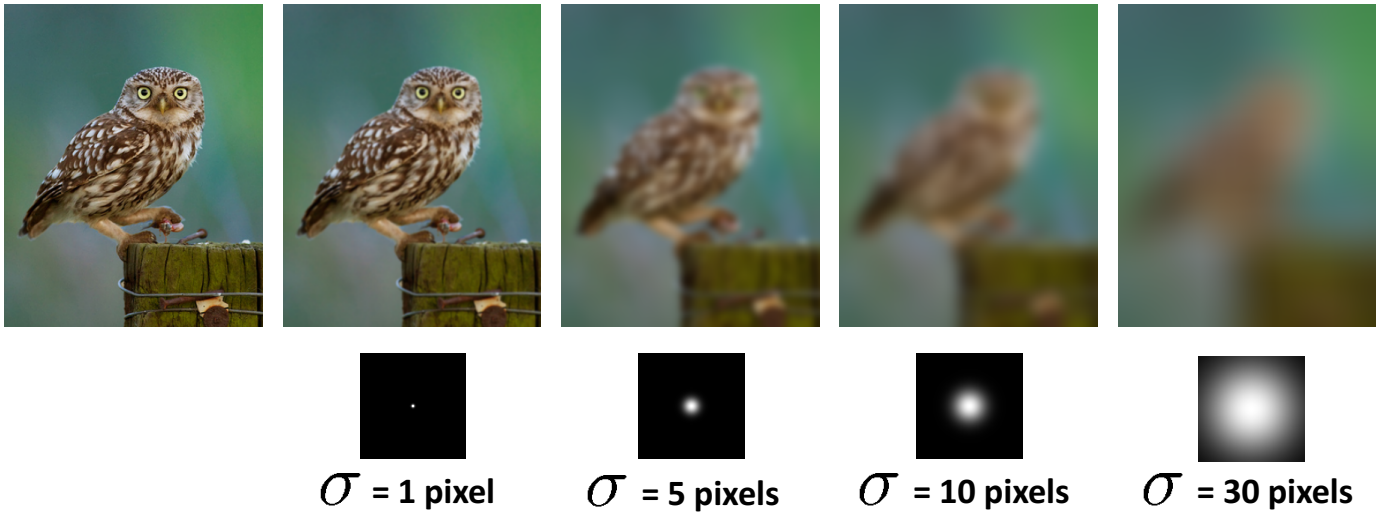
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Approximated by: $\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Source: C. Rasmussen

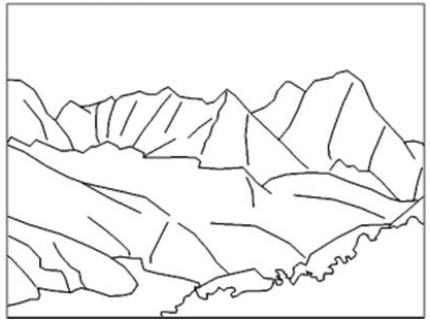
Gaussian filter



Edge Detection



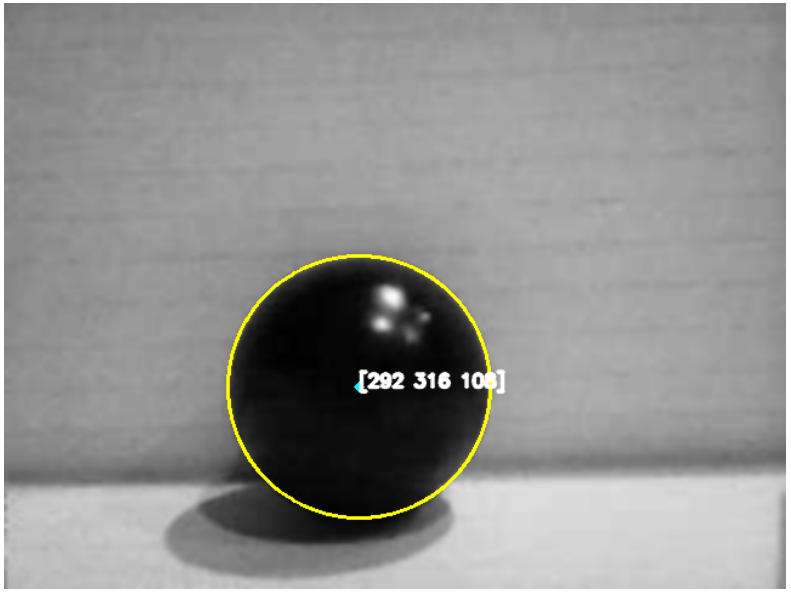





Edge Detection

How do we extract edges of a particular shape?

Straight Lines



Circles

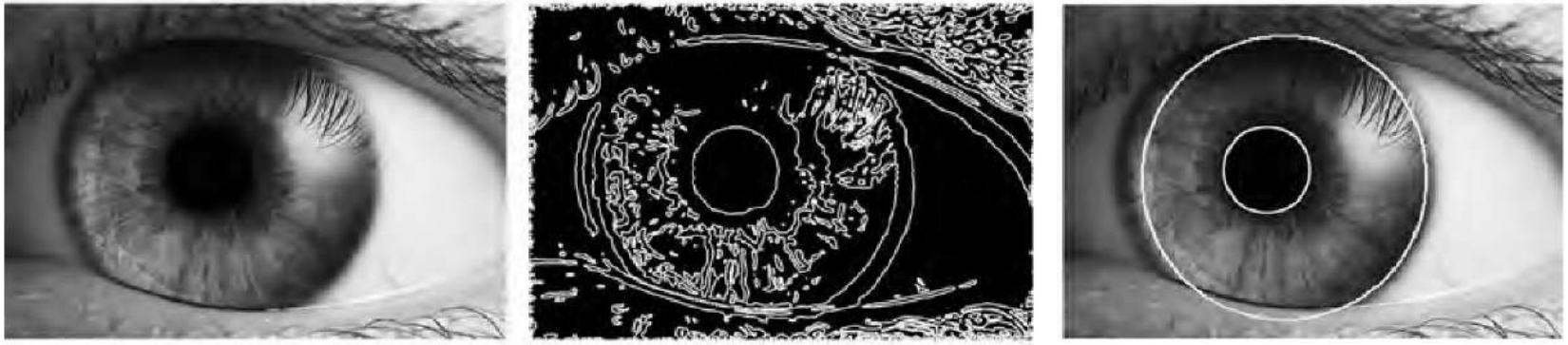
Hough Transform

- Performed after Edge Detection
- A technique to isolate the curves of a given shape in a given image
- Can locate regular curves like straight lines, circles, parabolas, ellipses, etc.
  - Requires that the curve be specified in some parametric form
- Key Idea: Edges **VOTE** for the possible model

Advantages of Hough Transform

- Tolerant of gaps in the edges
- It is relatively unaffected by noise
- It is also unaffected by occlusion in the image (the complete object does not need to be visible)

Examples



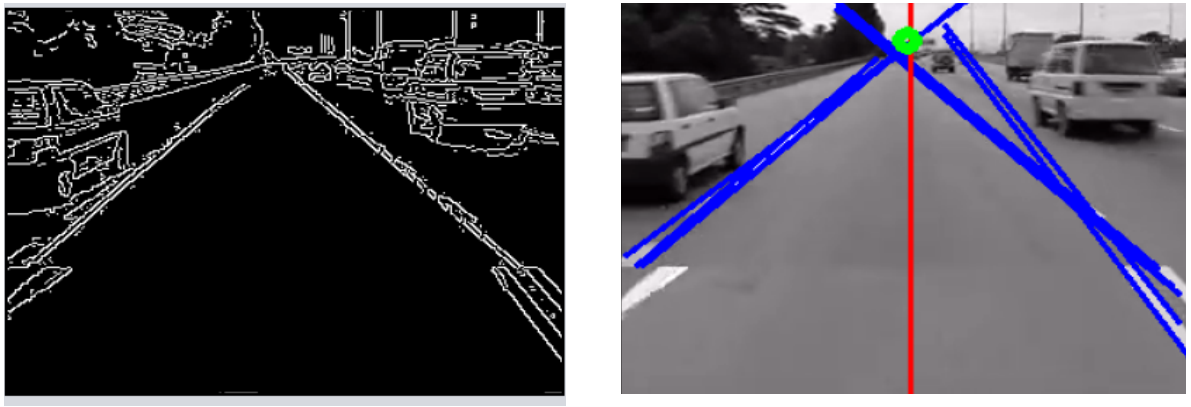


Image and Parameter Spaces

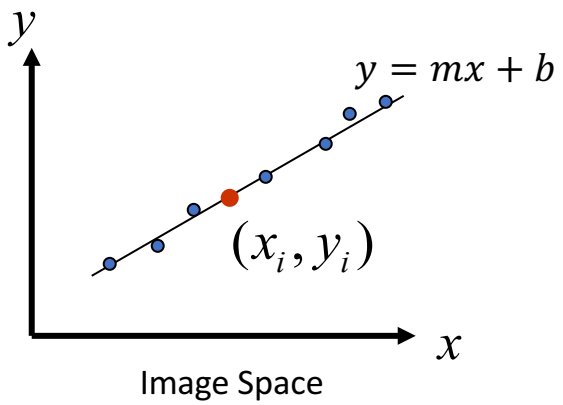
Equation of Line: $y = mx + b$

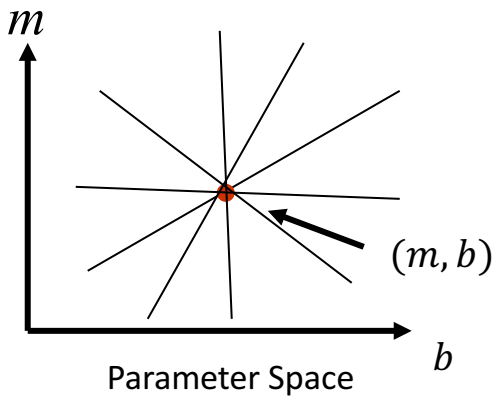
Find: (m, b)

Consider point: (x_i, y_i)

$$y_i = mx_i + b$$

Map the (m, b) coordinate into parameter space (also called Hough Space)





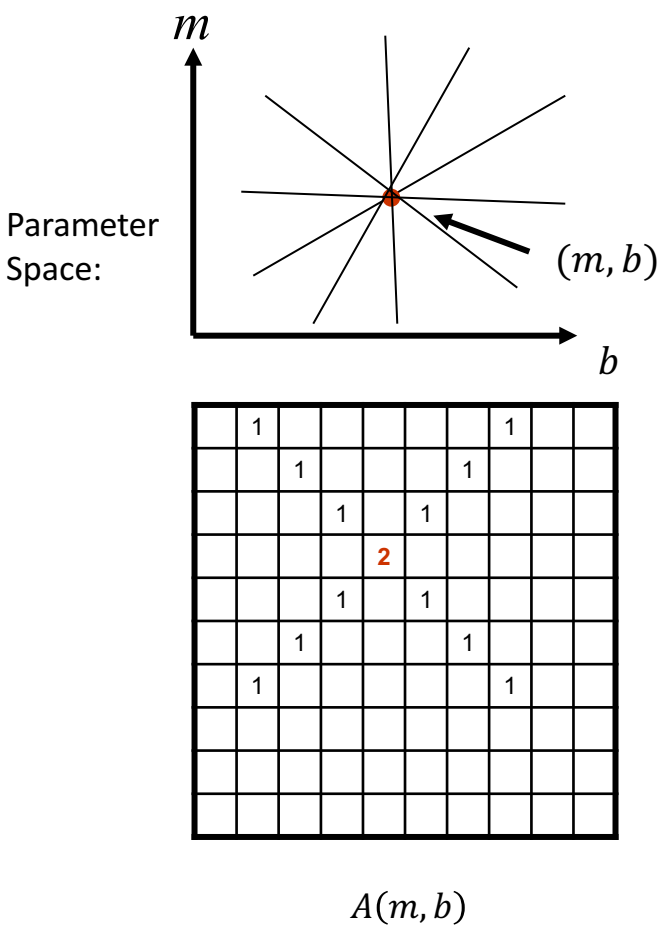
Line Detection by Hough Transform

1. Create 2D accumulator array A that discretizes parameters (m, b)
2. Initialize A to zero for all values of m and b
3. For each **edge** pixel (x_i, y_i):

   For all values of (m, b) where $y_i = mx_i + b$

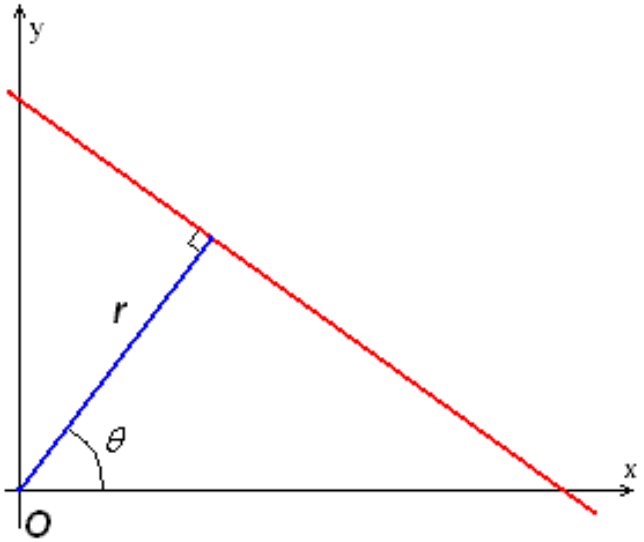
   $$A(m, b) \mathrel{+}= 1$$

4. Return (m, b) corresponding to the highest value in A



Line Representation

- The y = mx + b representation fails in case of vertical lines, so in reality we want to use polar coordinates r, θ



Single line, no noise

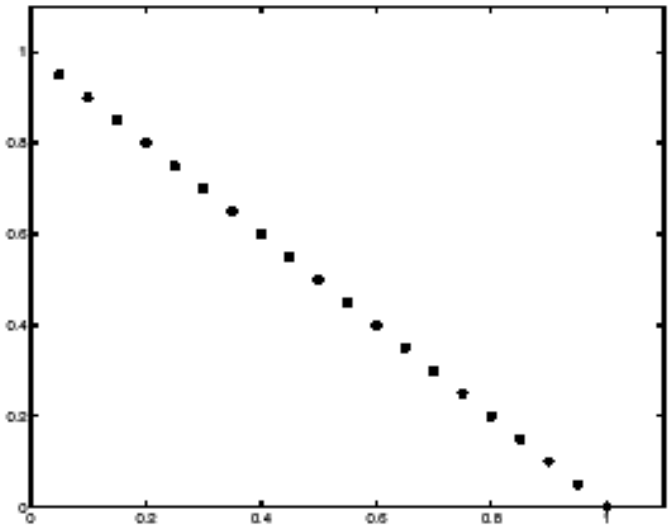
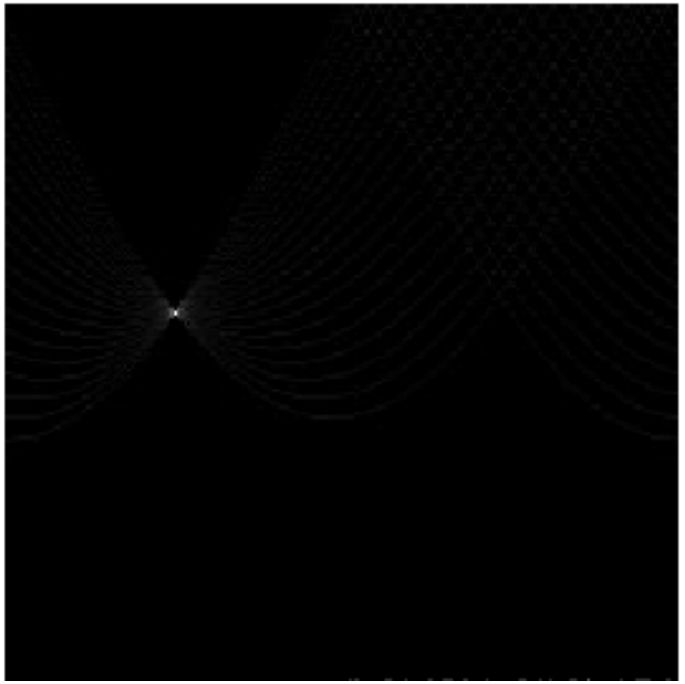


Image space



A

(using polar coordinates with horizontal axis as θ and vertical as r)

Single line with noise

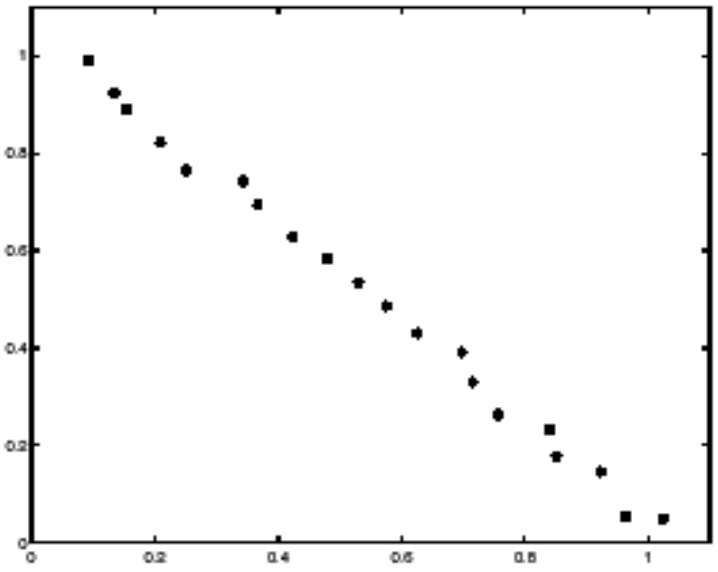
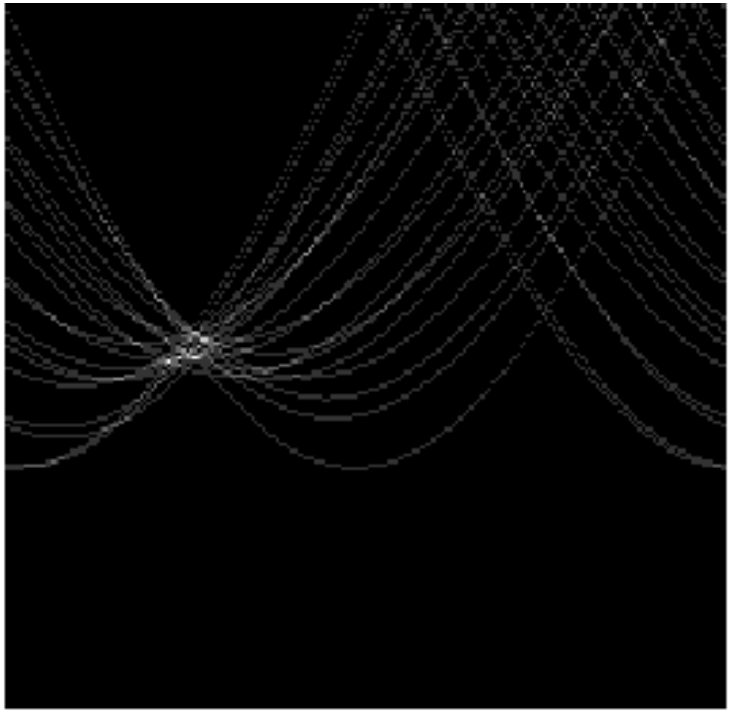


Image space



A

(using polar coordinates with horizontal axis as θ and vertical as r)

No Line

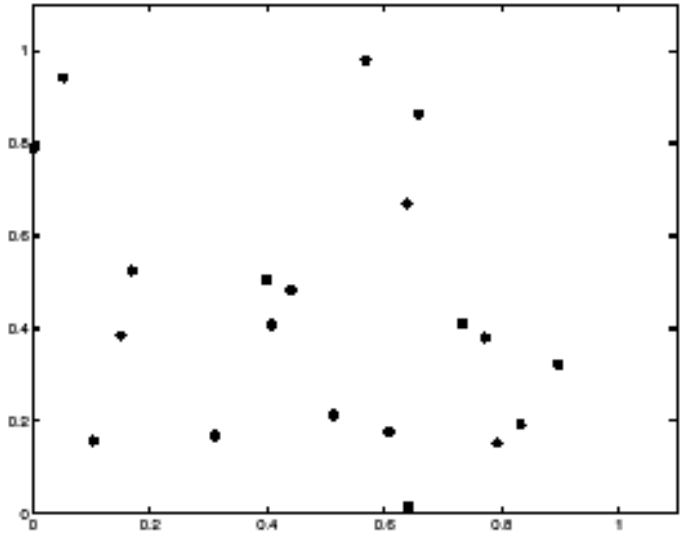
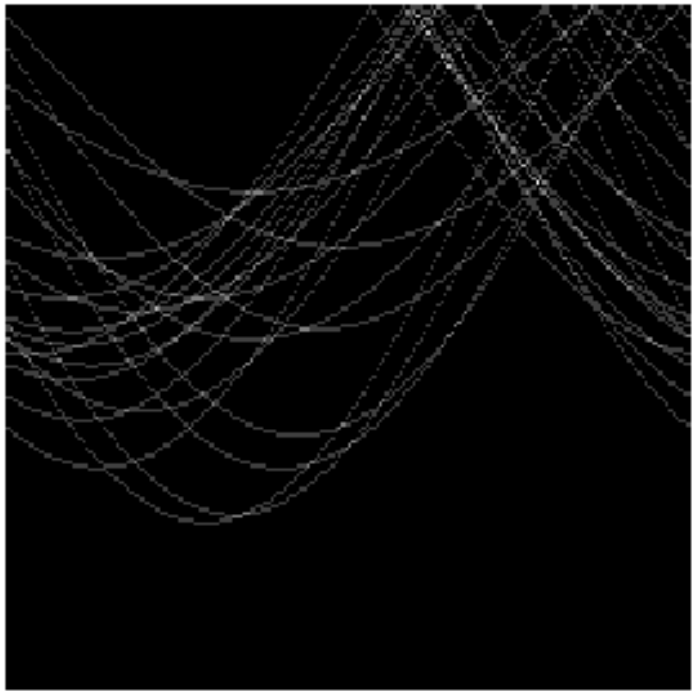


Image space



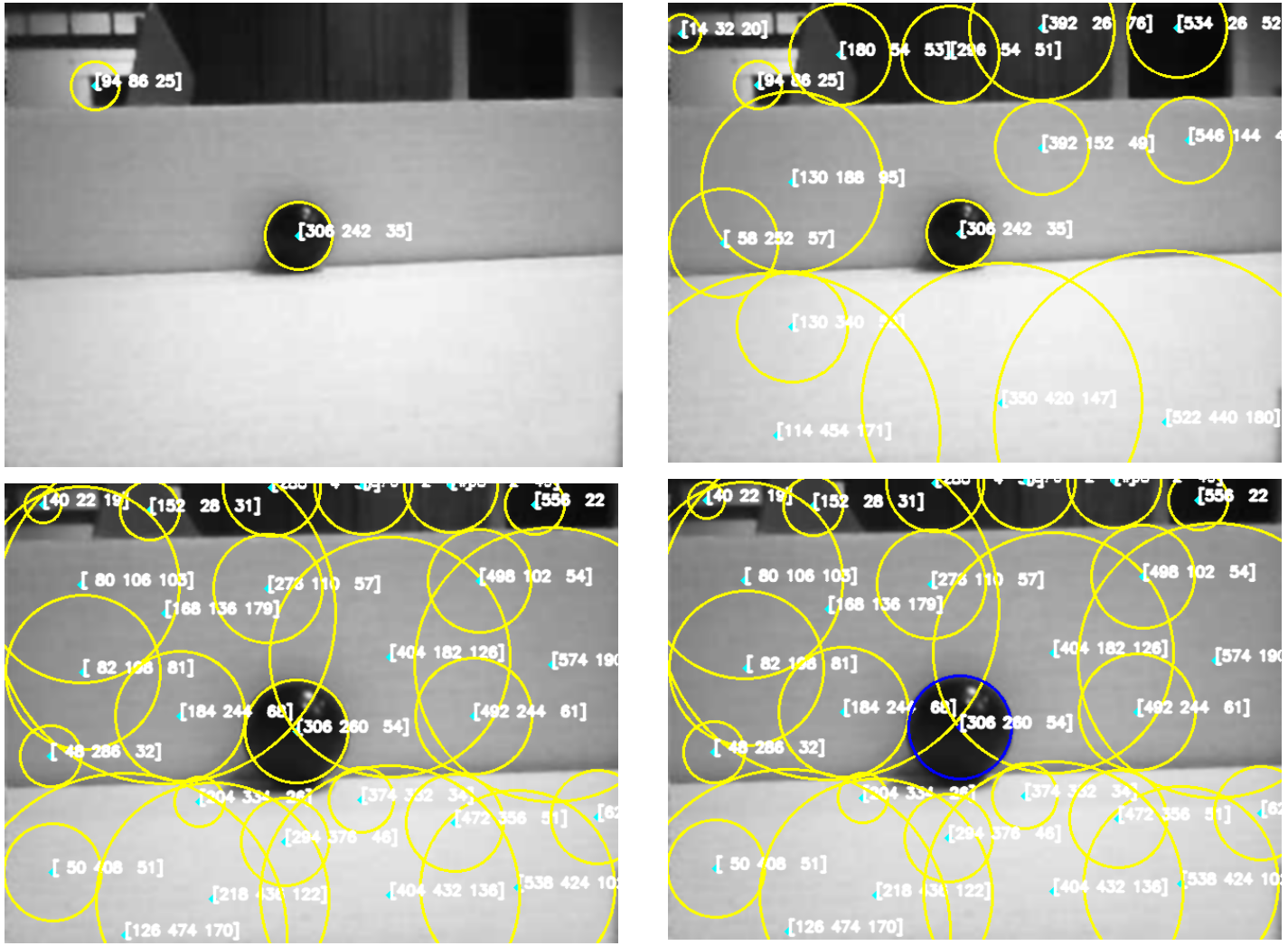
A

(using polar coordinates with horizontal axis as θ and vertical as r)

Same principle can be applied to circles (or any other shape)

- OpenCV has a built it HoughLines() and HoughCircles() functions

- For HoughCircles(), the code first applies the Canny edge detector, then finds circle centers and finally the best radius for each center.
- There are some additional implementation details we won't go into, such as using a gradient instead of keeping track of a 3D accumulator matrix A (which would be inefficient)



HoughCircles() has a lot of parameters that significantly affect the number and size of the candidate circles you get. If you choose to use this method, spend some time exploring different parameter values.

Hough Circles

- In OpenCV, the returned list of circles is ranked by confidence value, although we are not given the confidence value itself. The first is circle may or not be the one you're looking for. Consider other processing techniques (average intensity value, color, etc) to select the best circle.

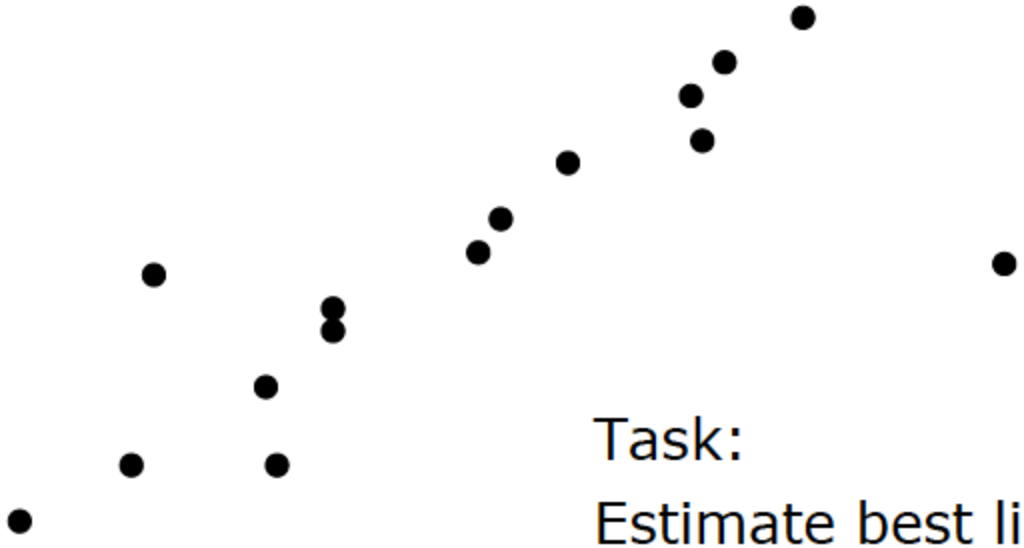
- Hough transform works well for line fitting but is hard to generalize to higher dimensions

- Another voting strategy: RANSAC

RANdom SAmple Concensus (RANSAC)

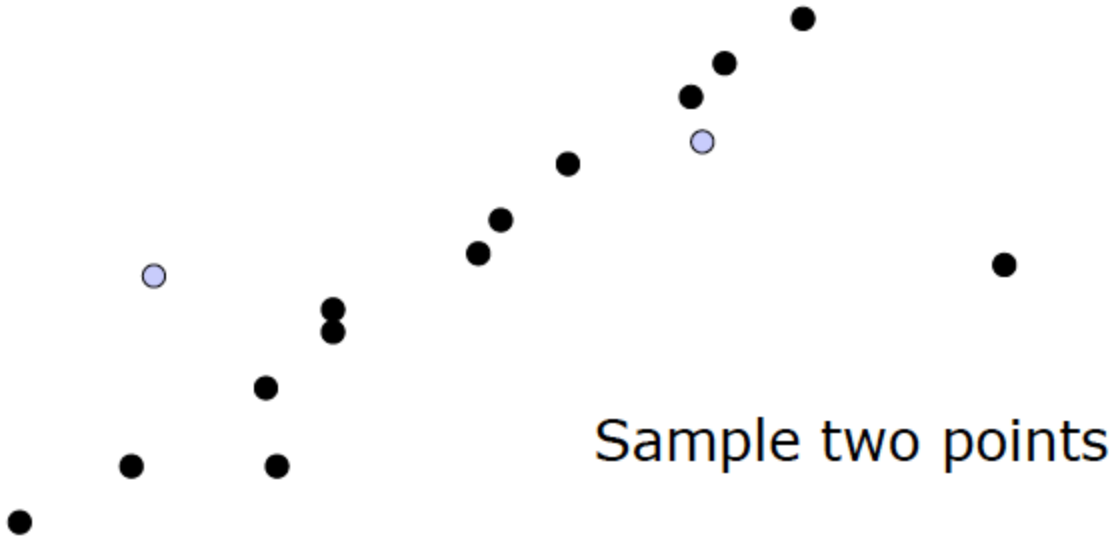
- Randomly choose s samples/points
  - Typically s = minimum sample size that lets you fit a model
- Fit a model (e.g., line) to those samples
- Count the number of inliers that approximately fit the model
- Repeat N times
- Choose the model that has the largest set of inliers*

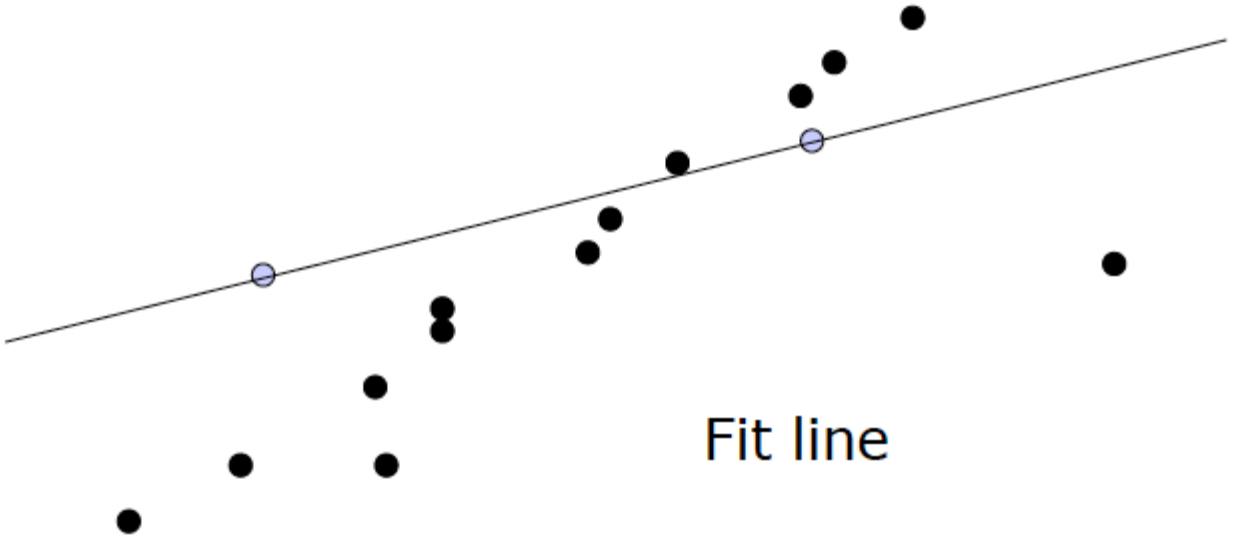
*better results are often achieved by using all the inliers from this step to create an updated model.

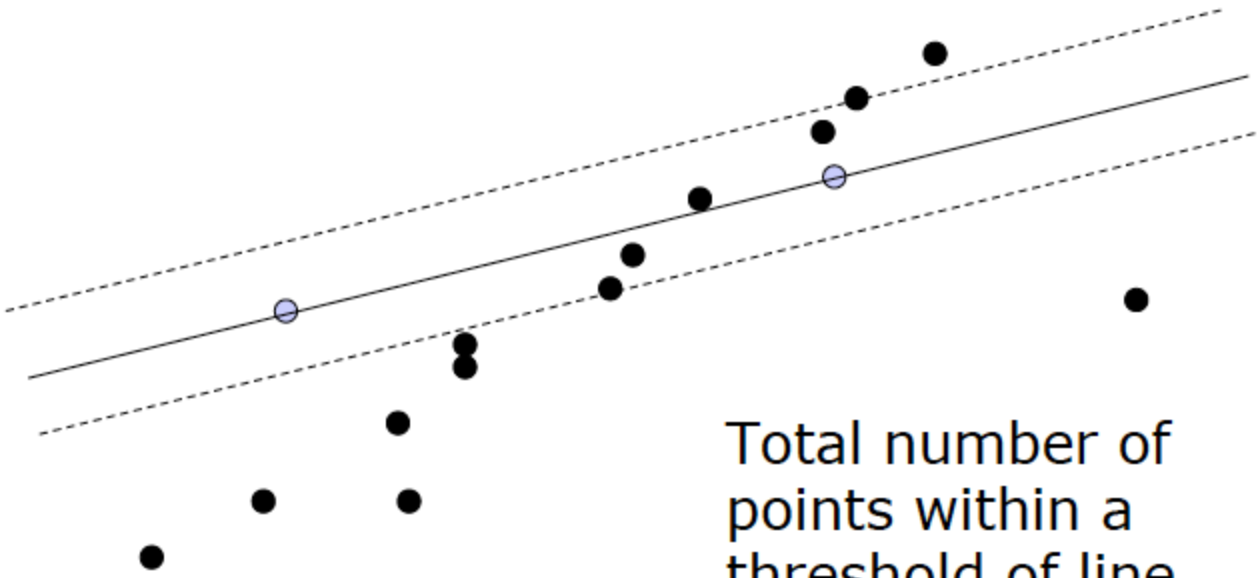


Task:

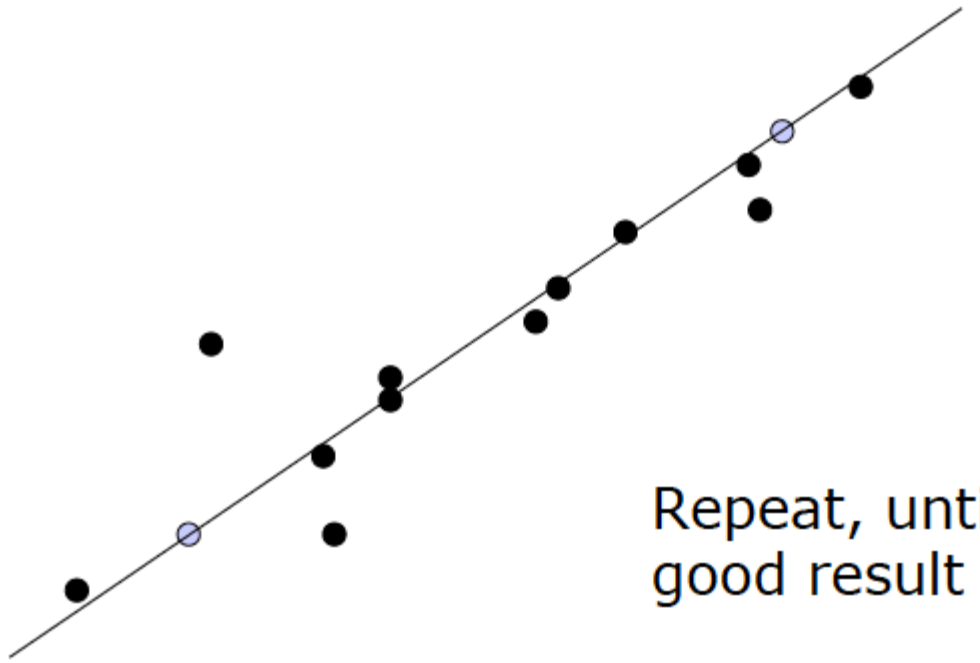
Estimate best line


Sample two points

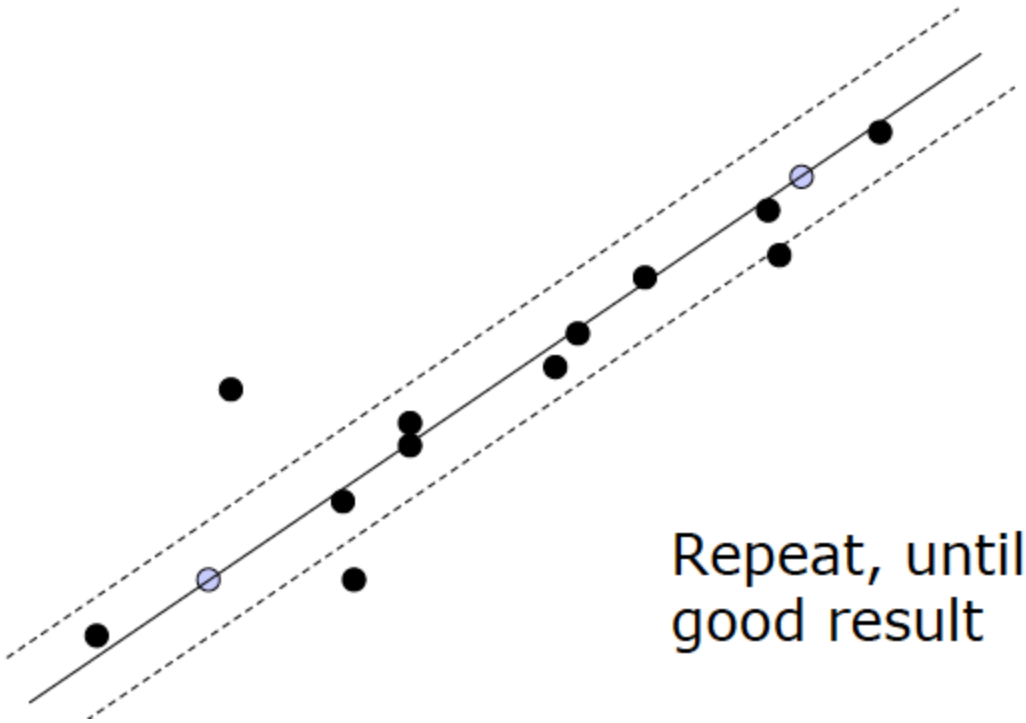

Fit line


Total number of
points within a
threshold of line


Repeat, until get a
good result



Repeat, until get a good result



Repeat, until get a good result

RANSAC parameters

- **Inlier threshold** related to the amount of noise we expect in inliers

- **Number of rounds** related to the percentage of outliers we expect, and the probability of success we'd like to guarantee

Suppose we know that 20% of the points are outliers, and we want to fit the correct line with 99% probability. How many rounds do we need?

- Let w be the probability of selecting an inlier (.8 in this example)
- We need $n = 2$ points to fit a model (line)
- What is the probability of selecting $n = 2$ inliers to create a good model? $w^2 = .8^2 = 0.64$
- Then the probability of selecting points that result in a bad model is $1 - w^2 = .36$

- If we run N iterations of RANSAC and want the correct answer with some probability p (99% above):

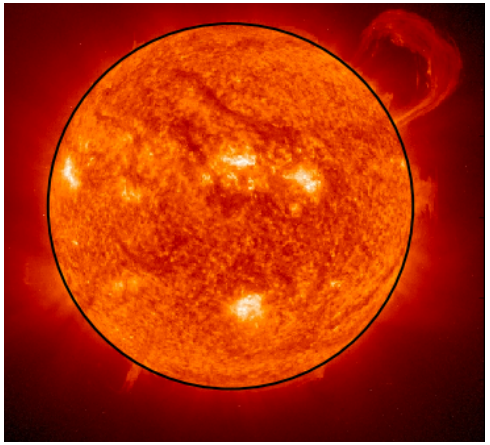
$$(1 - w^n)^N = 1 - p$$

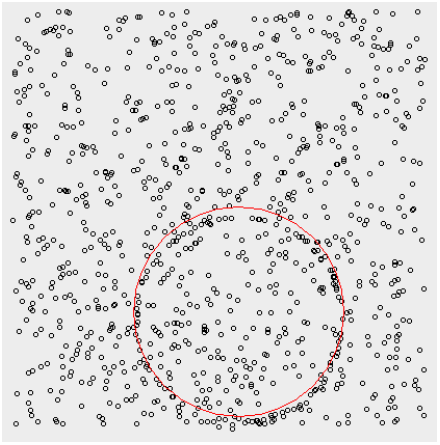
- $N = \frac{\log(1-p)}{\log(1-w^n)}$

- $N = \frac{\log(1-.99)}{\log(1-.8^2)} = 4.5$

RANSAC

- RANSAC works extremely well for both low and high-dimensional problems, including circle fitting





Other considerations

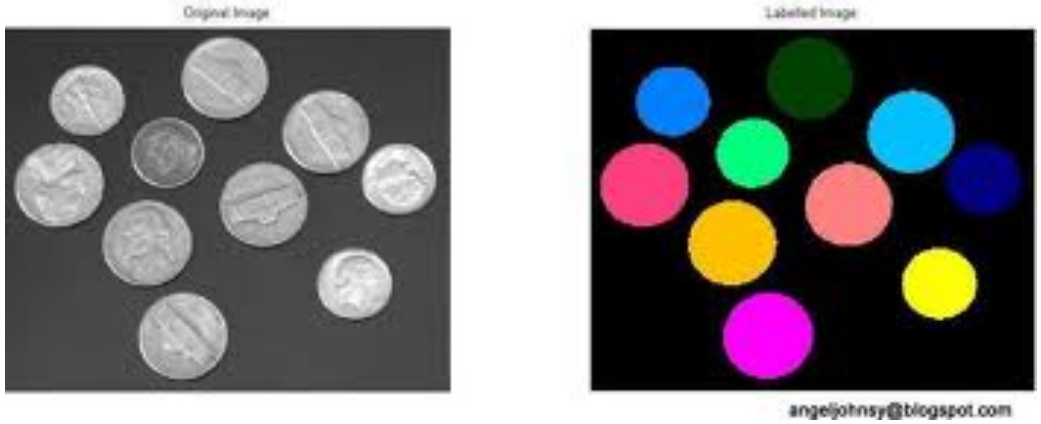
- Noise comes in many forms and will affect all of the above methods to varying degrees.
  Smoothing/blurring can reduce noise, but caution should be used to avoid blurring away the edges.

- Feel free to explore other helpful OpenCV features, nothing is off limits!

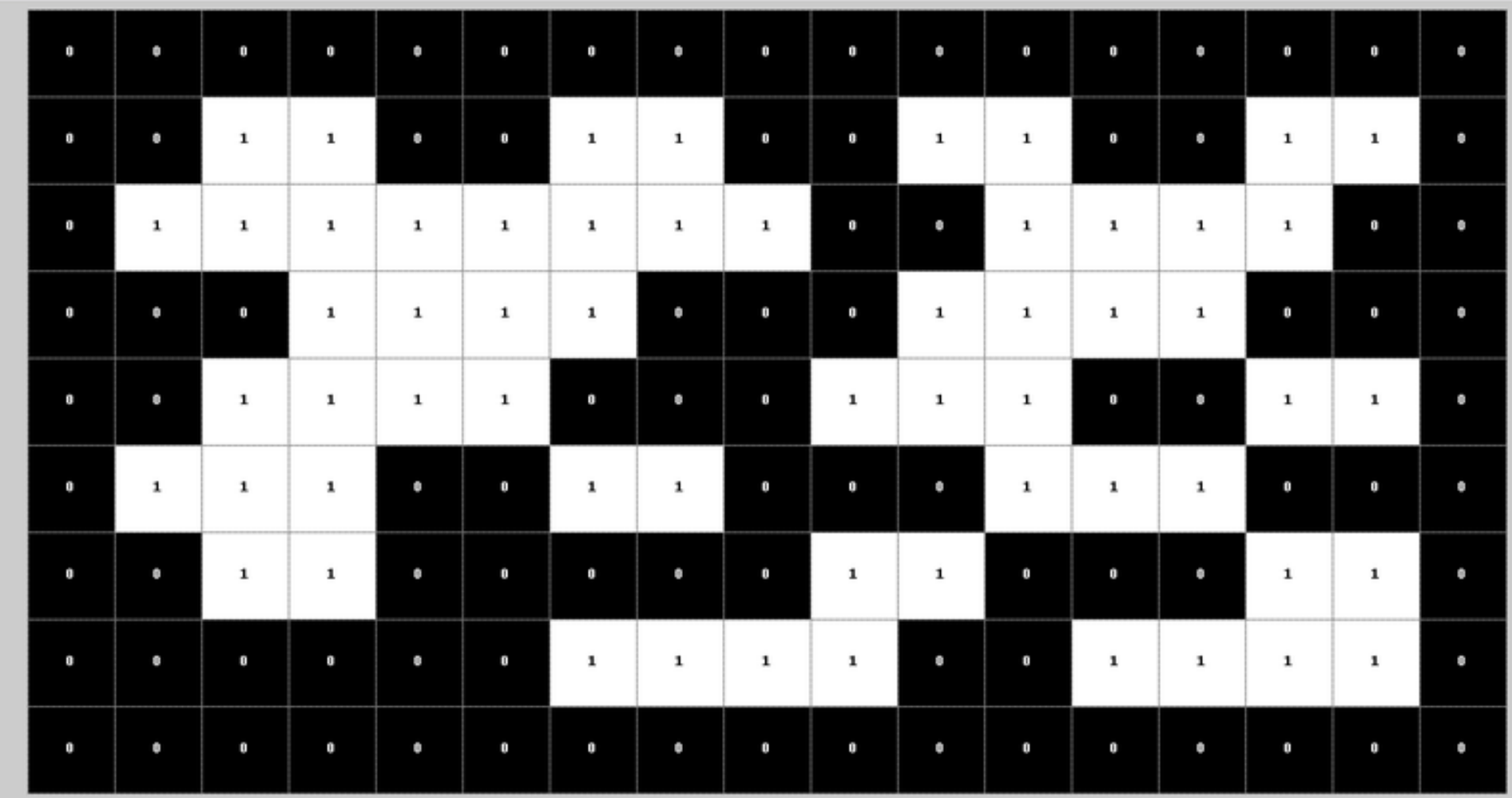
Connected-component labeling

Connected-component labeling

- A.k.a. connected-component analysis, blob extraction, region labeling, blob discovery, or region extraction
- algorithmic application of graph theory, where subsets of connected components are uniquely labeled

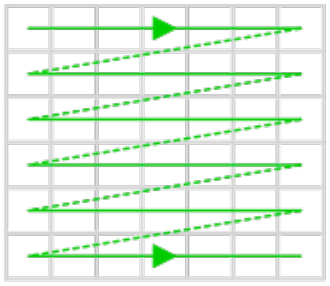


File Edit Window Help

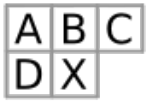


Pixel info: (X, Y) Intensity

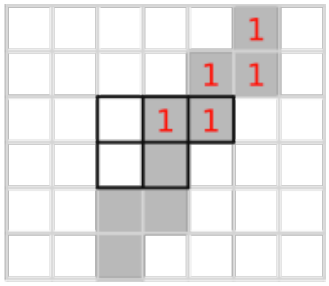
Two-Pass Algorithm



Iterate through pixels
from top left to bottom right

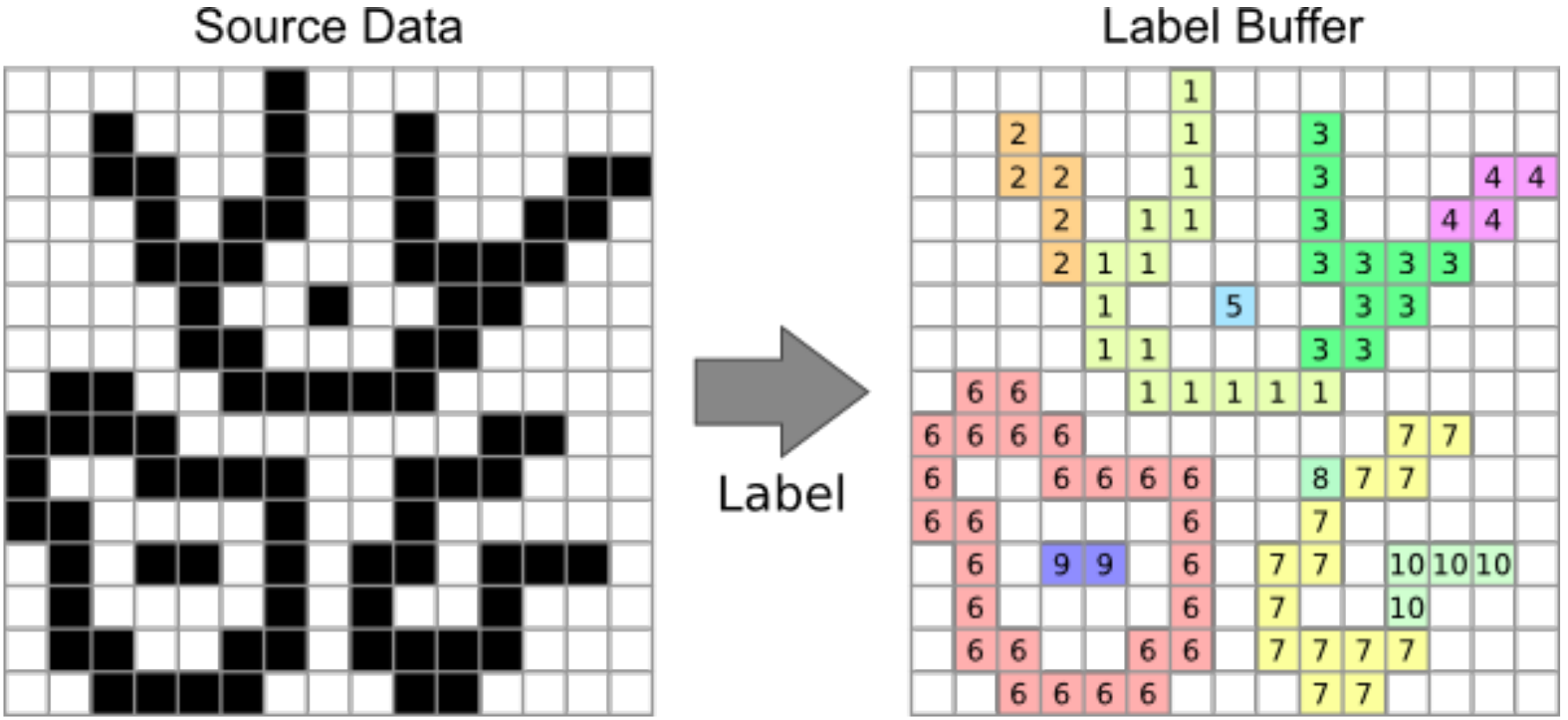


Label Kernel
Applied to each pixel



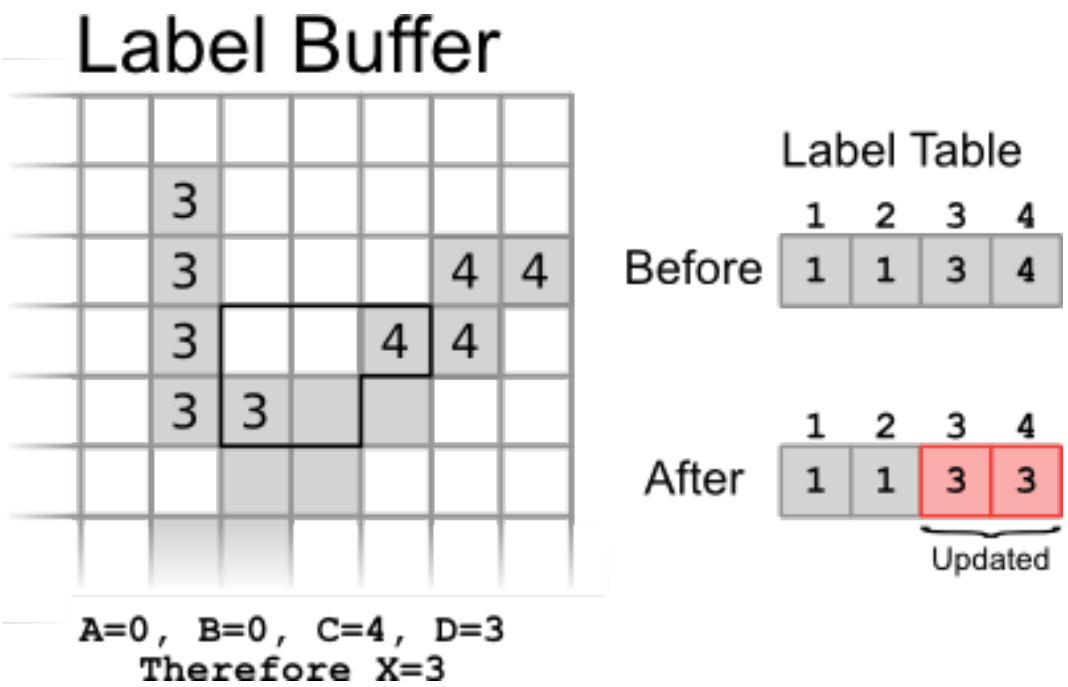
A=0, B=1, C=1, D=0
Therefore X=1

```
l = 1                                   // Initial Label number
for each pixel
  if pixel X is foreground
    if neighbours A,B,C & D are unlabelled (equal to zero)
      label pixel X with l
      increment l
    else
       num = neighbour label A,B,C & D with least value, not including 0
       label pixel X and pixels A, B, C & D if foreground with num
    end if
  end if
done
```

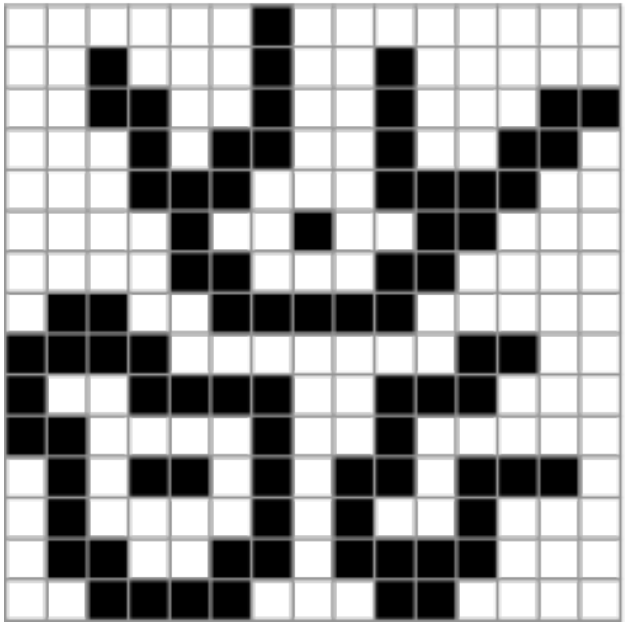


Label Table

1	2	3	4	5	6	7	8	9	10
1	1	1	3	5	6	7	7	9	7


Label Buffer
3
3
4 4
3
4 4
3 3
A=0, B=0, C=4, D=3
Therefore X=3
Label Table
1 2 3 4
Before
1 1 3 4
1 2 3 4
After
1 1 3 3
Updated

Source Data



Label

Label Buffer

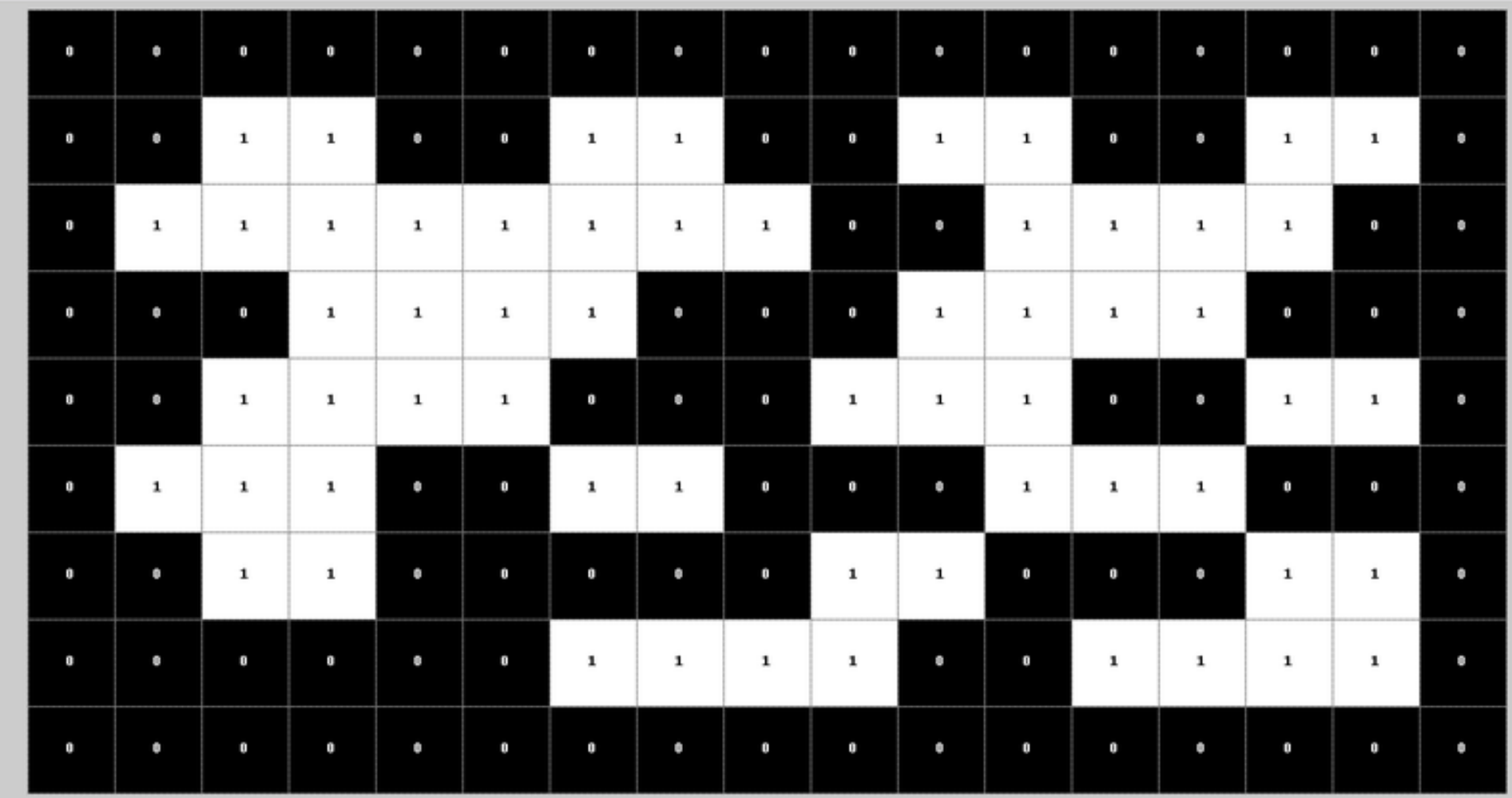
Label Table

1	2	3	4	5	6	7	8	9	10
1	1	1	3	5	6	7	7	9	7

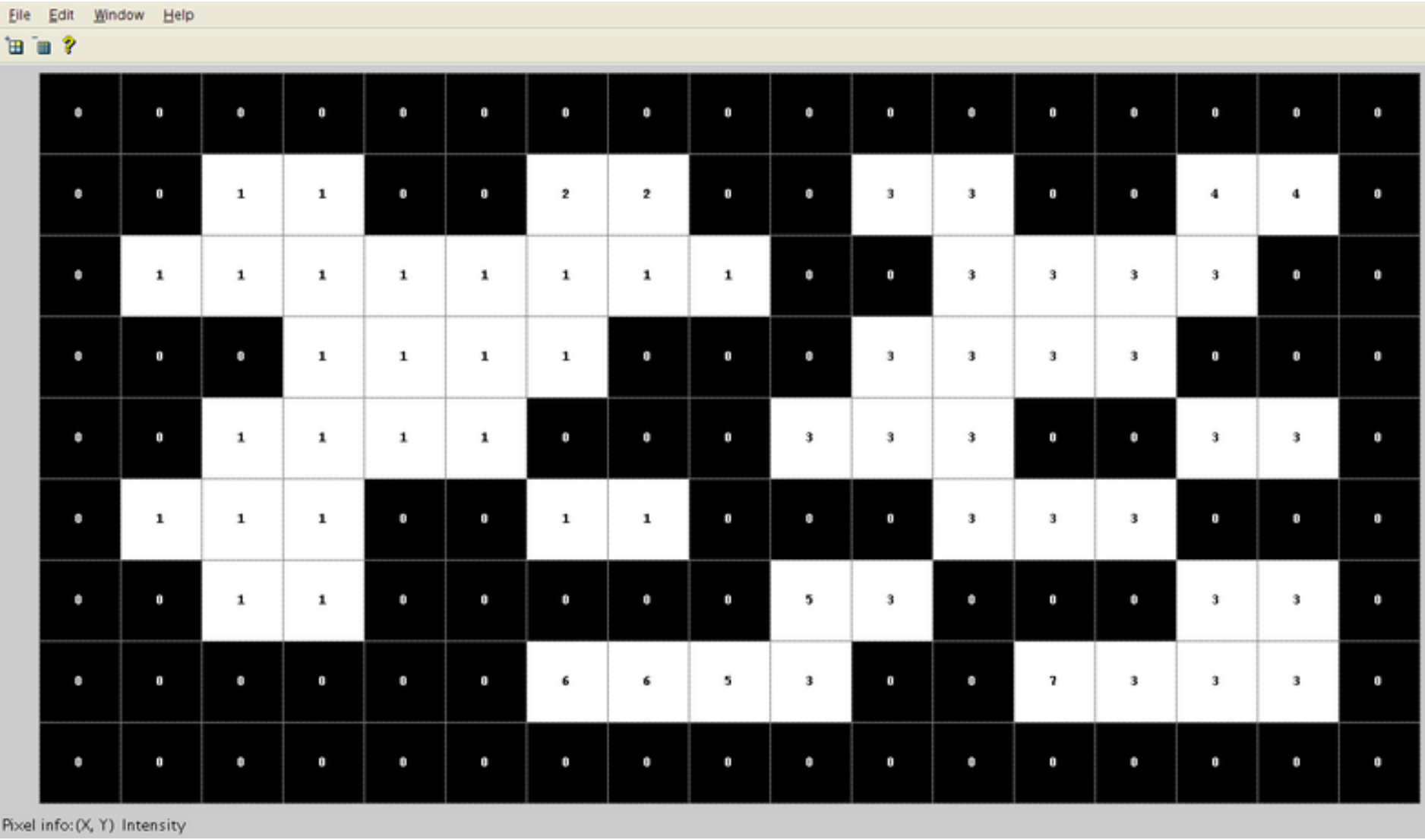
Blob	Labels
1	1, 2, 3 & 4
2	5
3	6
4	7, 8 & 10
5	9

Another example…

File Edit Window Help

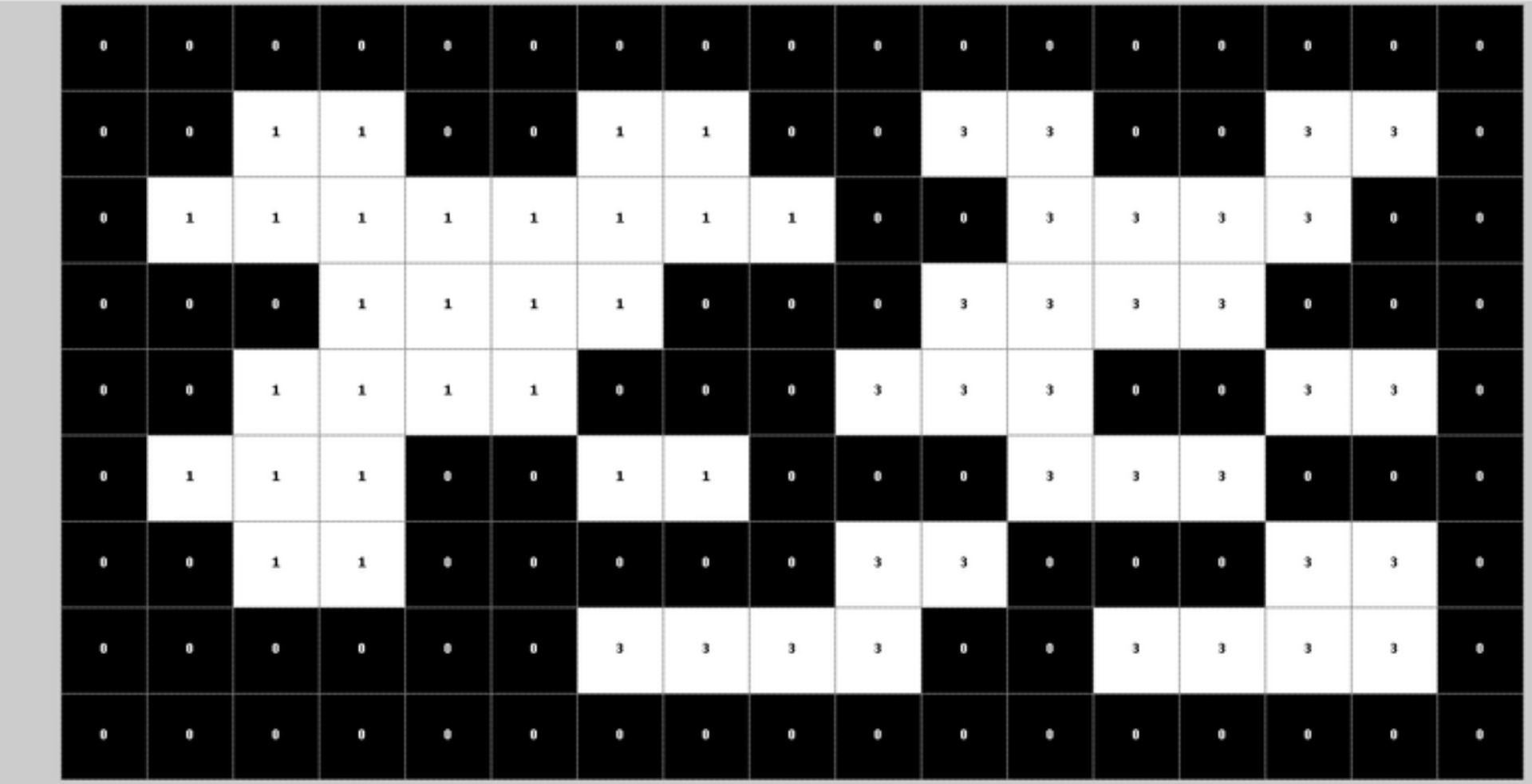


Pixel info: (X, Y) Intensity



First pass, 7 labels.

File Edit Window Help

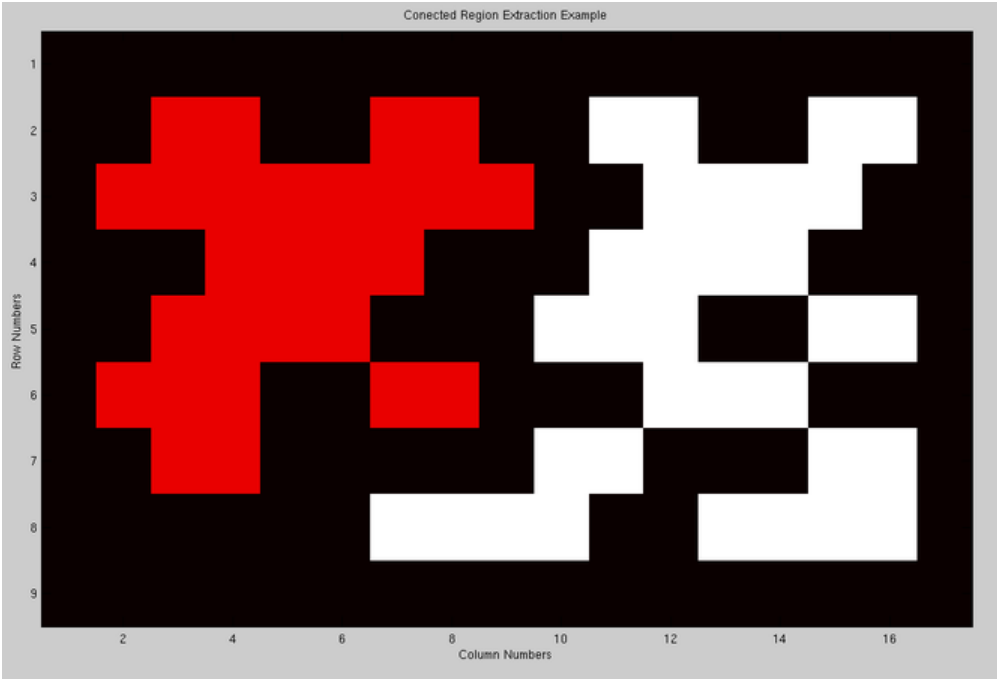


Pixel info:(1, 2) 0

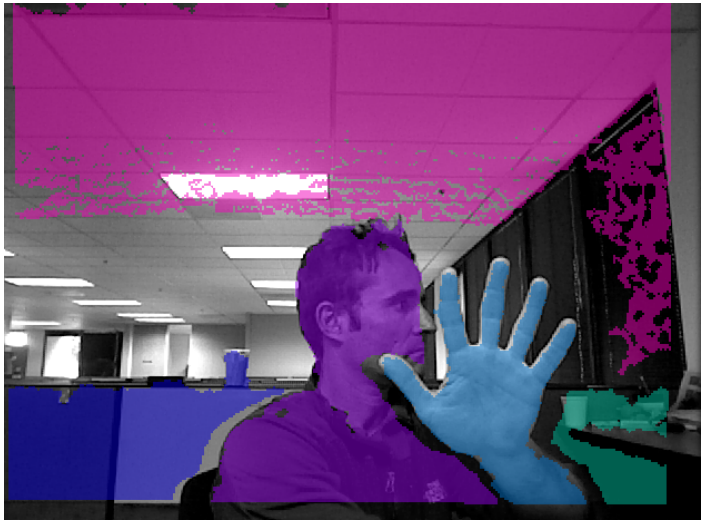
Following merging

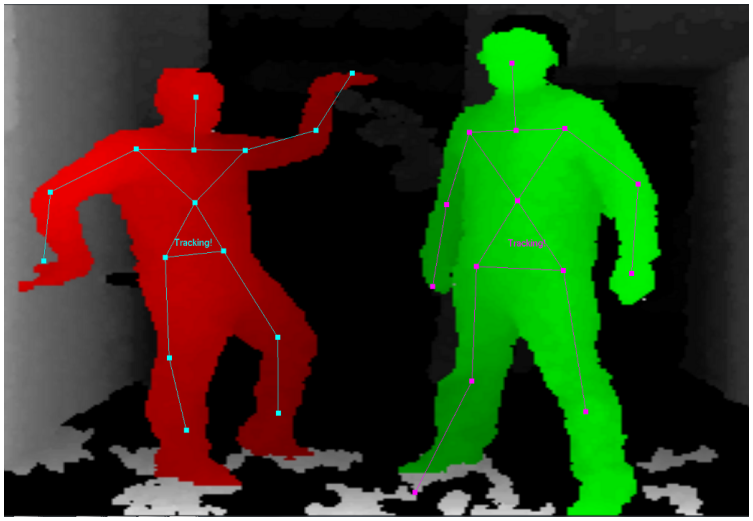
Set ID	Equivalent Labels
1	1,2
2	1,2
3	3,4,5,6,7
4	3,4,5,6,7
5	3,4,5,6,7
6	3,4,5,6,7
7	3,4,5,6,7

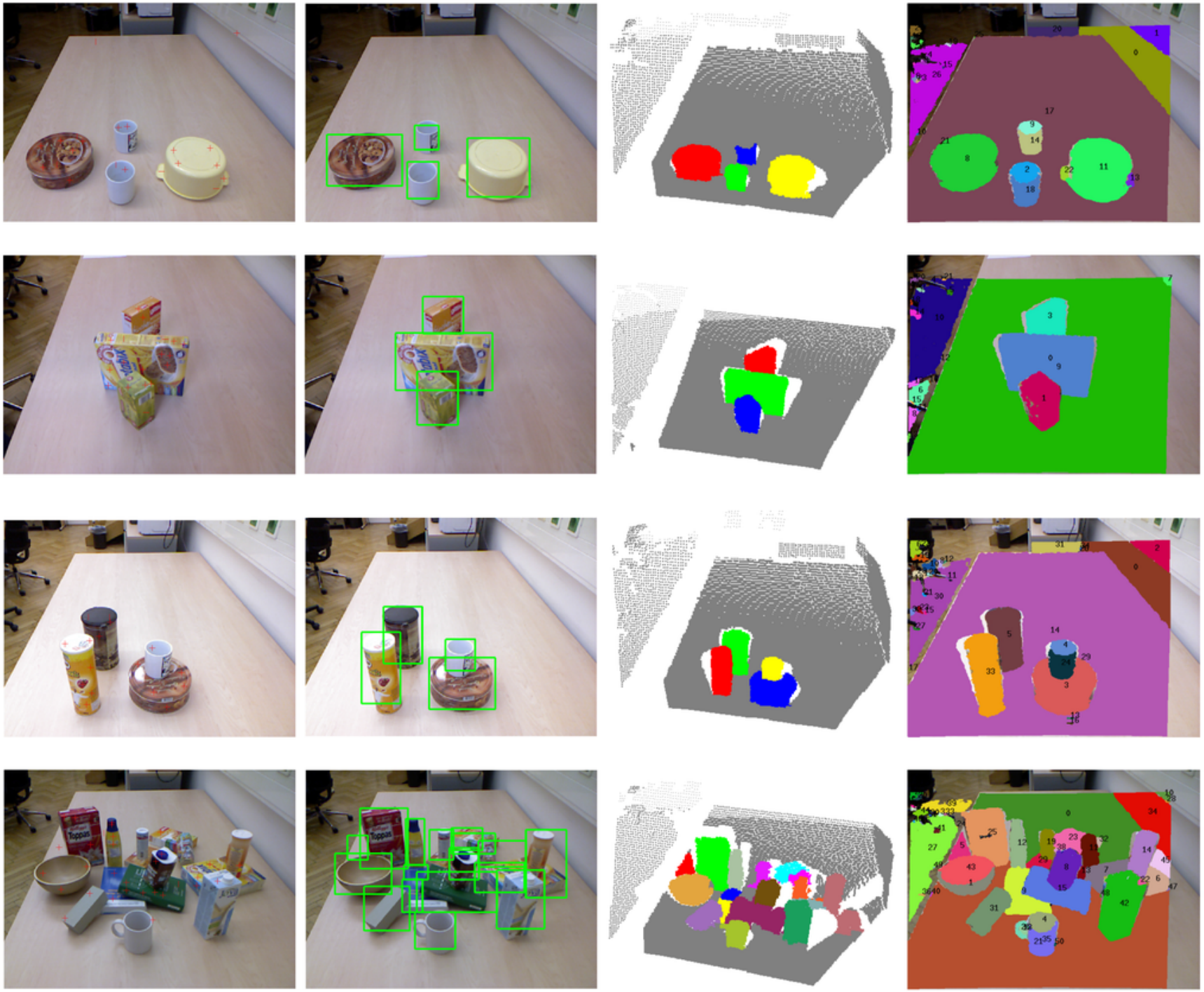
Equivalence classes for previous slide.



Final




Tracking!
Tracking!



References

- OpenCV Hough Transform documentation
- RANSAC tutorial
- OpenCV Connected Components documentation