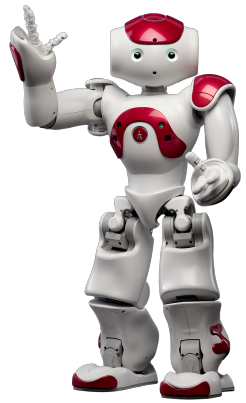
Image Processing

CS 3630 | Sonia Chernova

Primary sensor for most robotic platforms



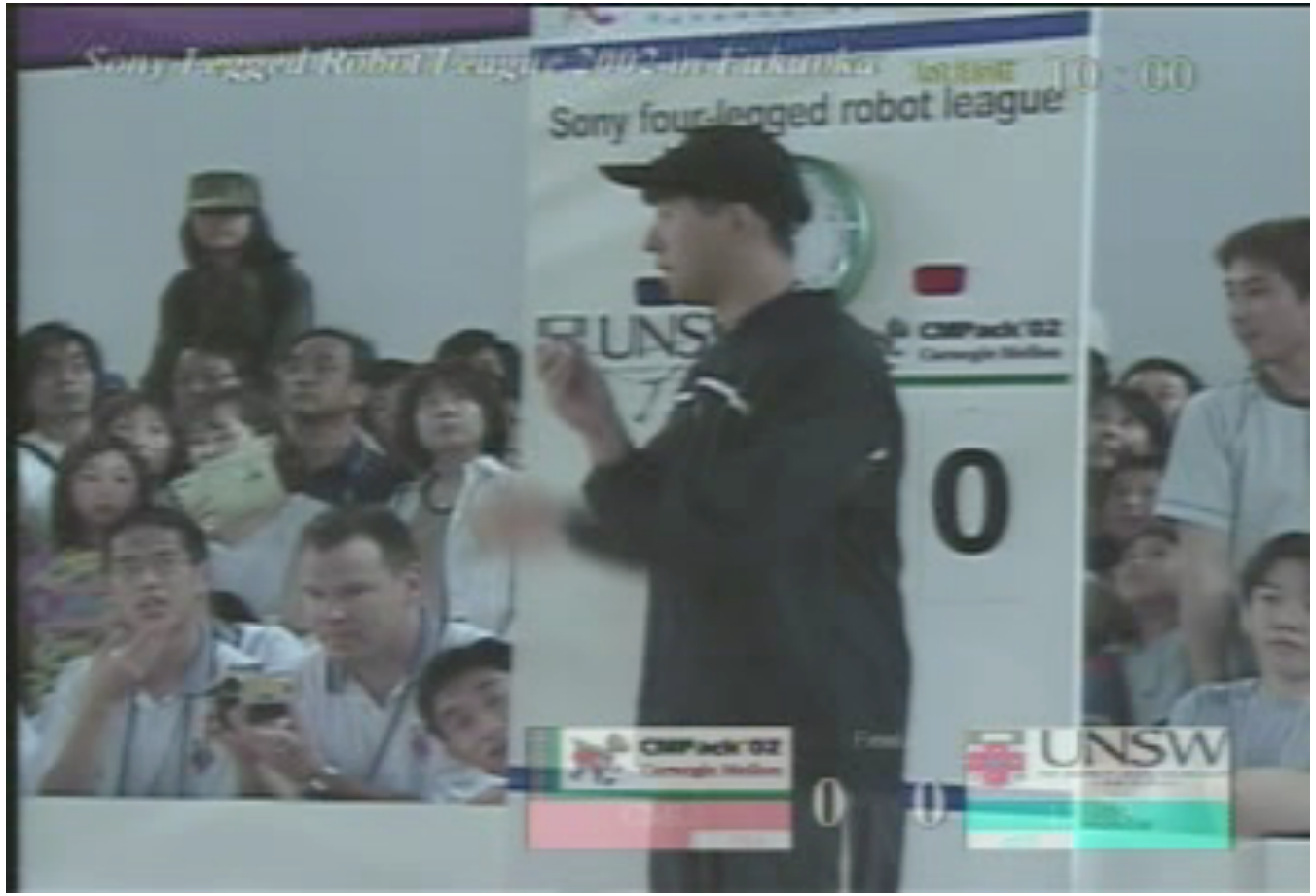
Common camera types

For now we'll focus here

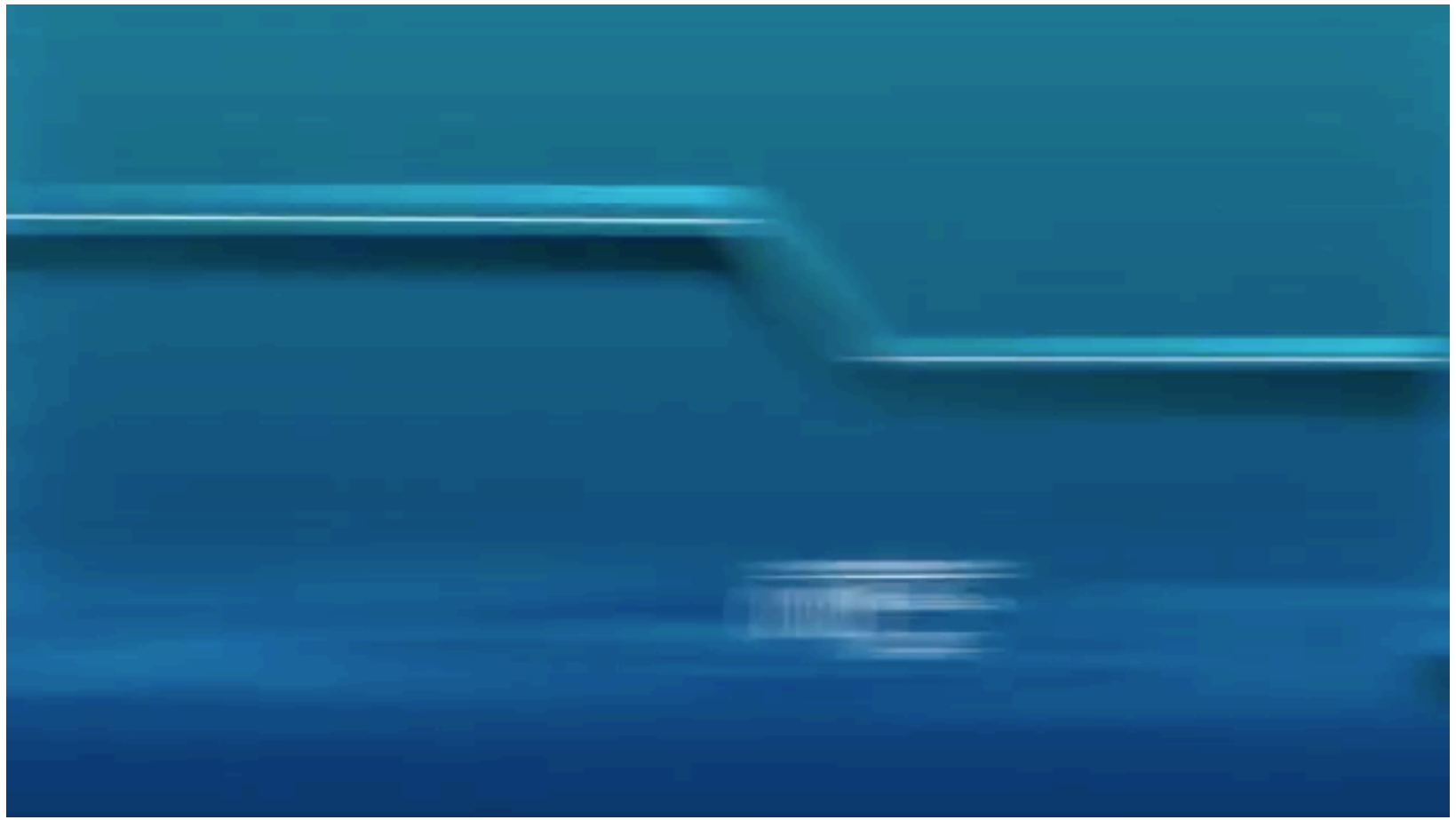
Single view
RGB image

Stereo
RGB-D image

Structured light or time of flight
RGB-D image


Sony four-legged robot league
0
UNSW
0
0


SONY
SONY
UNSW
3 3




Carnegie Mellon
RoboCup Vision Demo
www.cs.cmu.edu/~robosoccer

Images

- An image is basically a 2D array of intensity/color values
- Image types:

Color

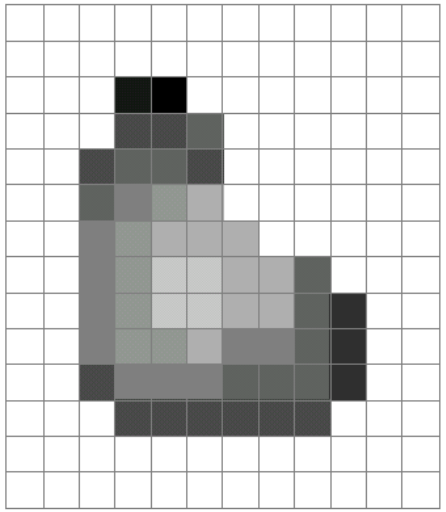
Grayscale

B-W

Cozmo images are grayscale

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)

Image sampling

640x640

320x320

160x160

80x80

40x40

20x20

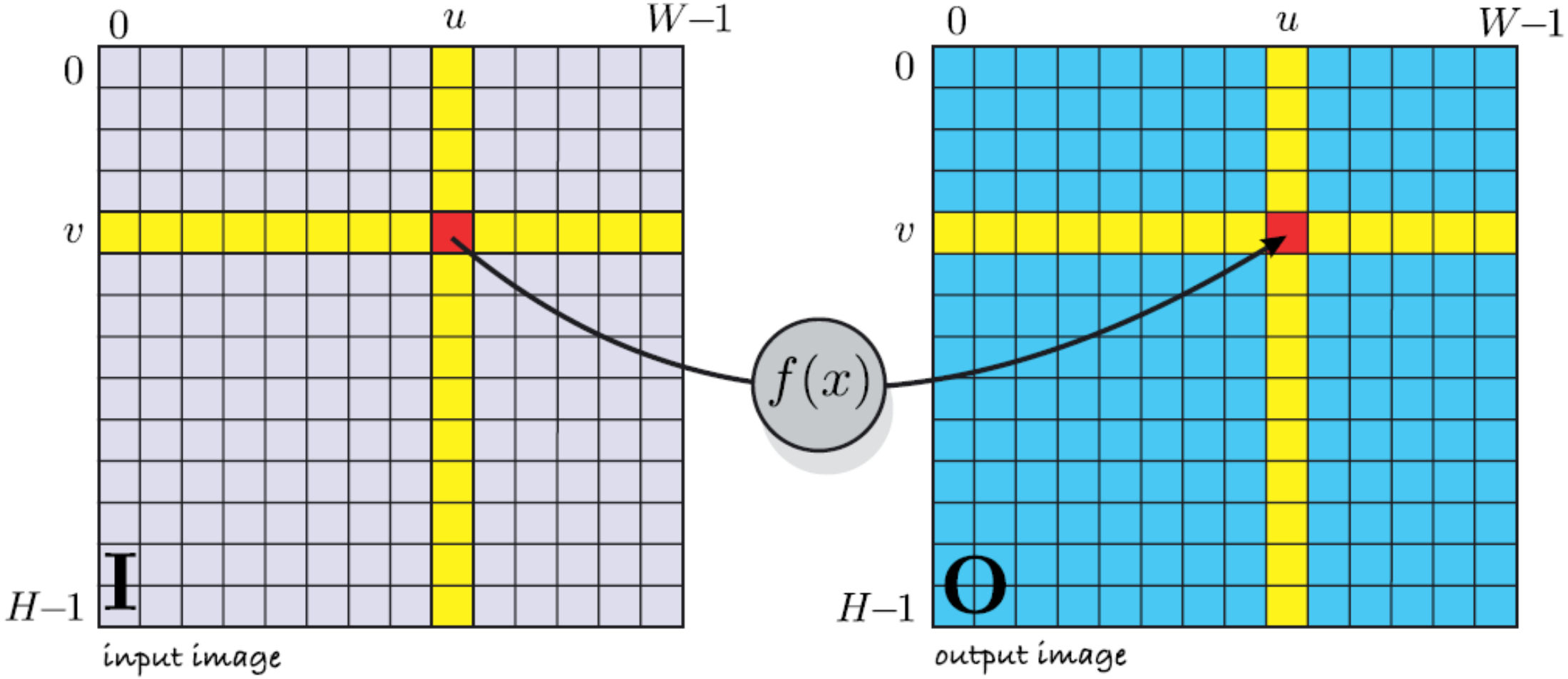
Filtering

- Filtering refers to:
  - Noise removal
  - Edge detection
  - Texture description
  - Multi-scale algorithms
  - Feature detection
  - Matched filters
  - …

- We will only focus on a few specific methods useful for our application

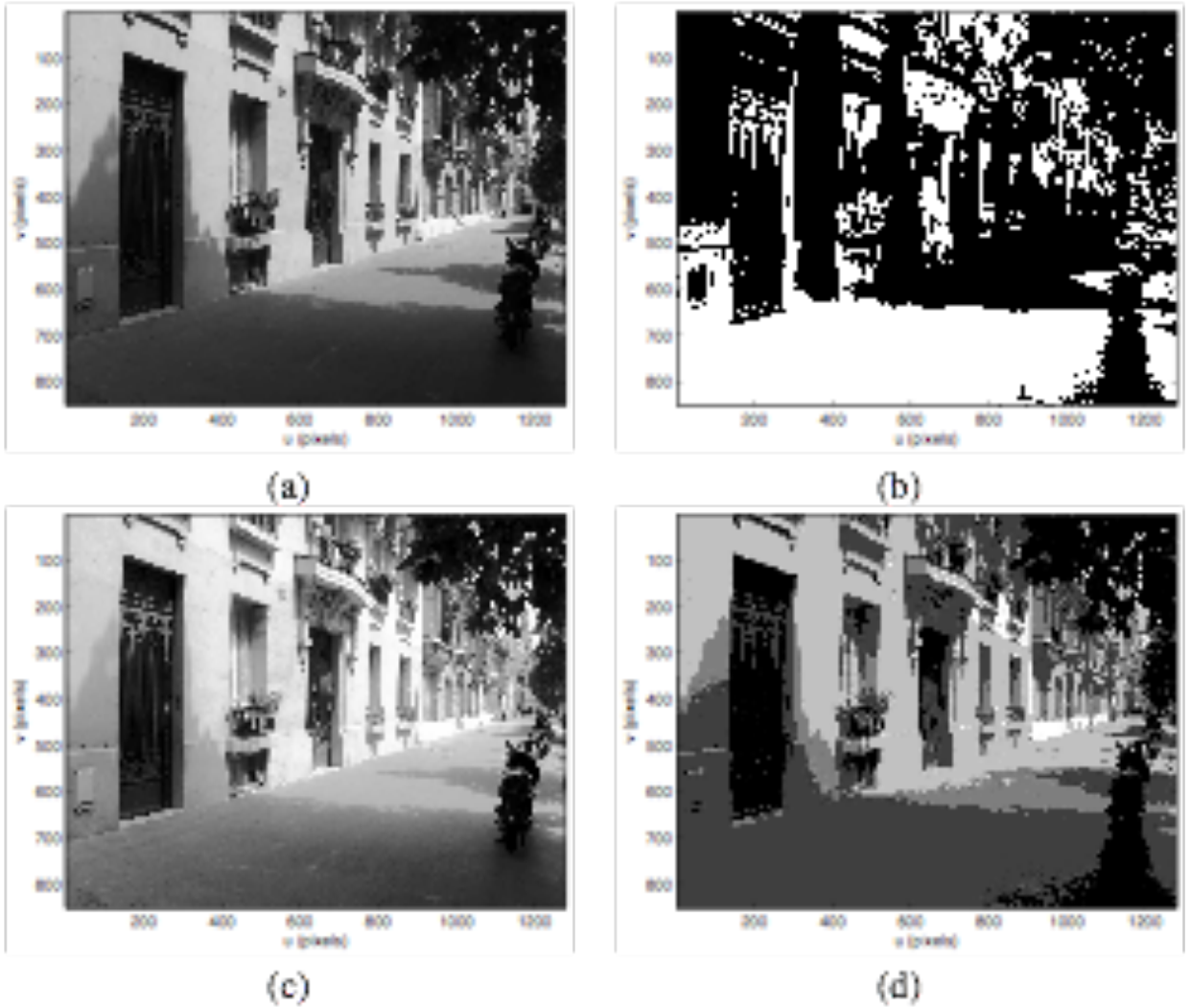
Monadic operators

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



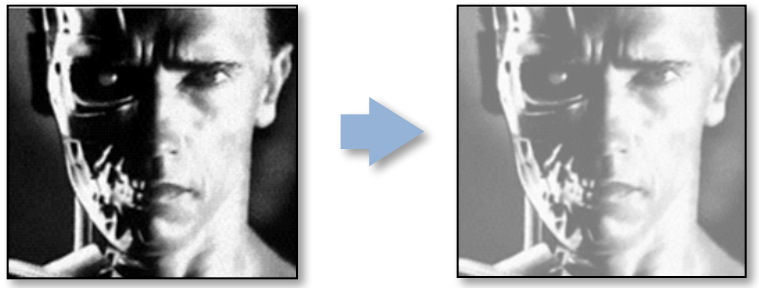
1-1 mapping between pixels

Signal enhancement

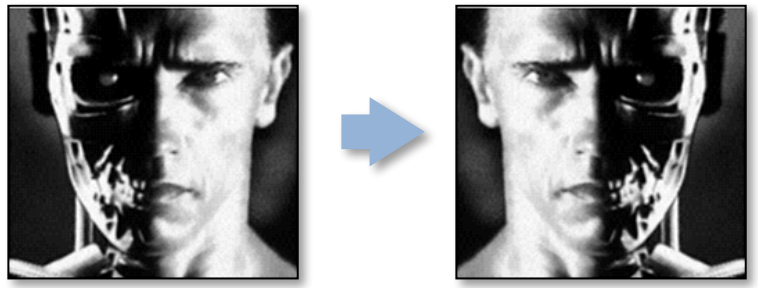


Some monadic image operations, **a** original, **b** shadow regions, **c** histogram normalized, **d** posterization

Image transformations



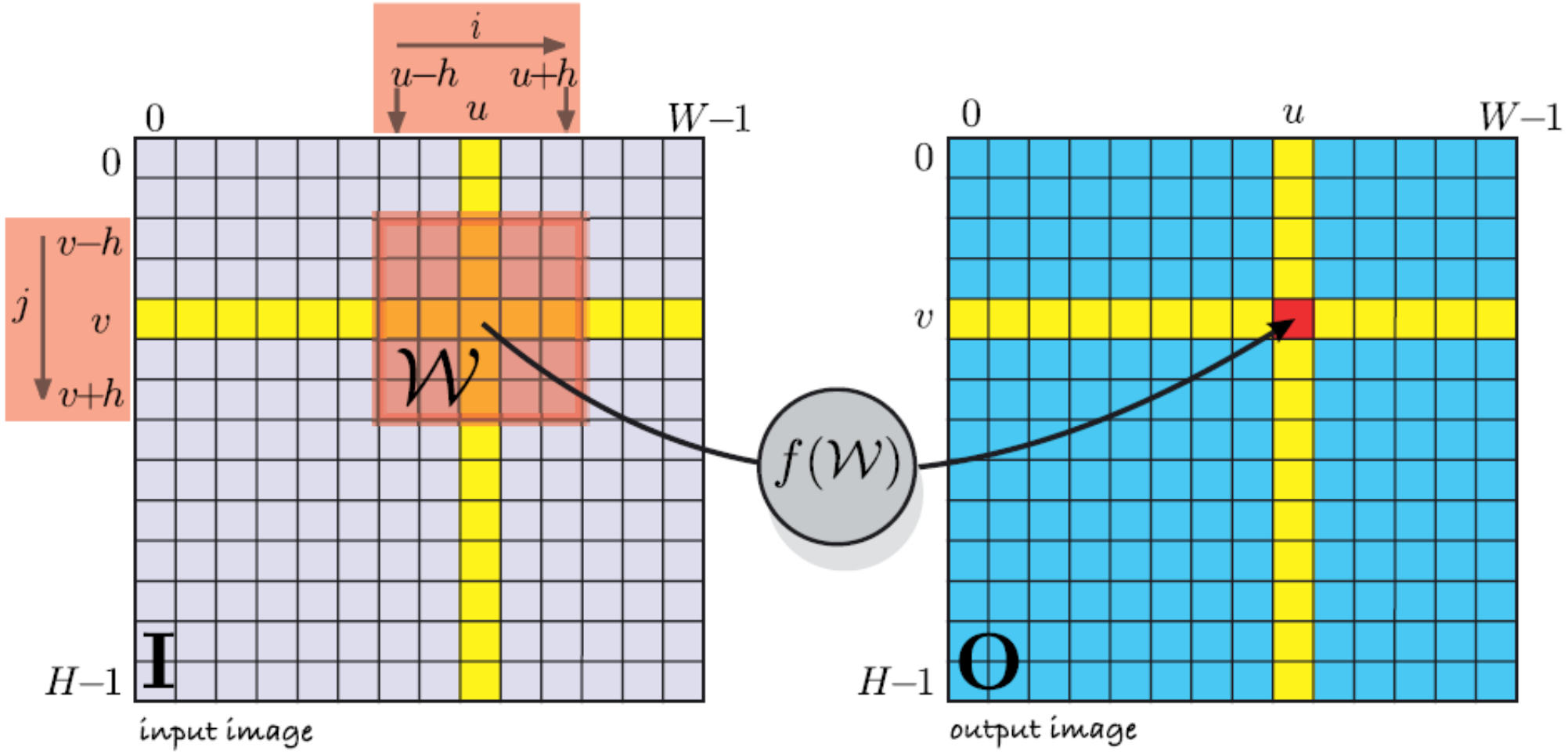
$$g(x,y) = f(x,y) + 20$$



$$g(x,y) = f(-x,y)$$

Local operators

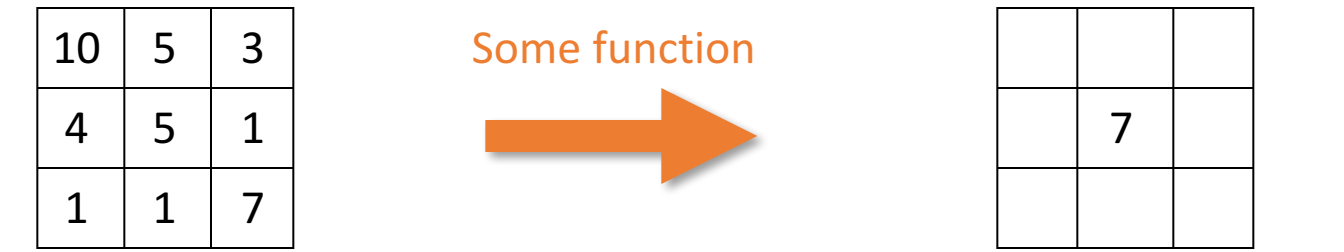
Local image processing operations. The reddish shaded region on the left shows the window W that is the set of pixels used to compute the red output pixel on the right



many-to-one mapping between pixels

Image filtering

- Modify the pixels in an image based on some function of a local neighborhood of each pixel



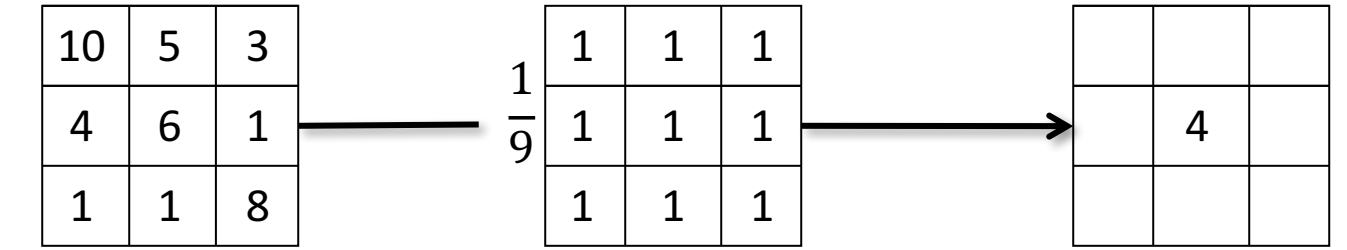
Local image data

Modified image data

Source: L. Zhang

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”

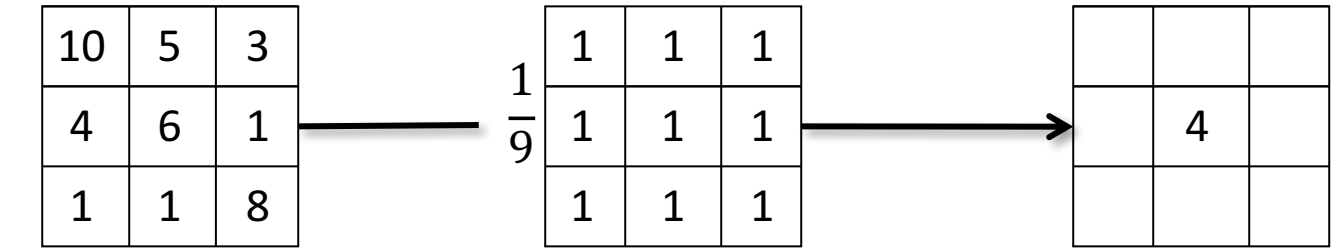


Source: L. Zhang

Linear filtering

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

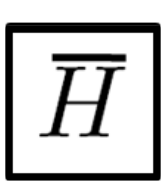
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

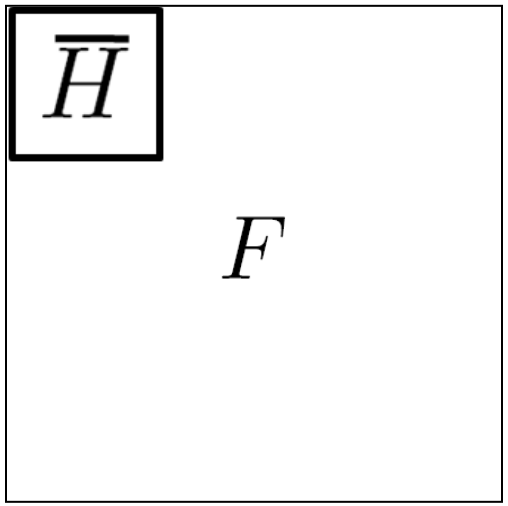


Source: L. Zhang

Convolution

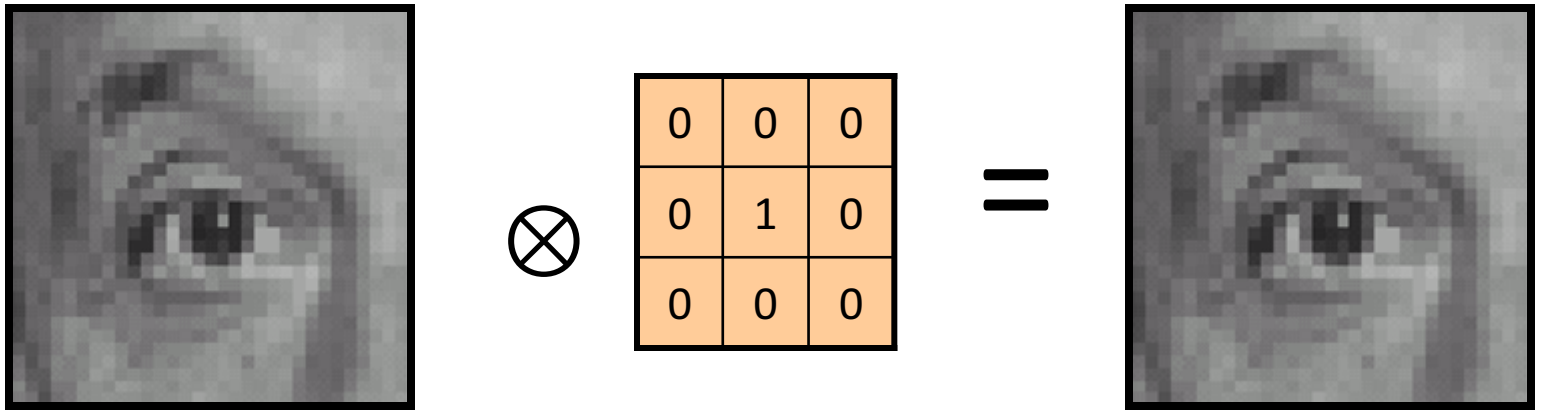
- Applying a kernel to an image in this way is called convolution





Adapted from F. Durand

Linear filters: examples

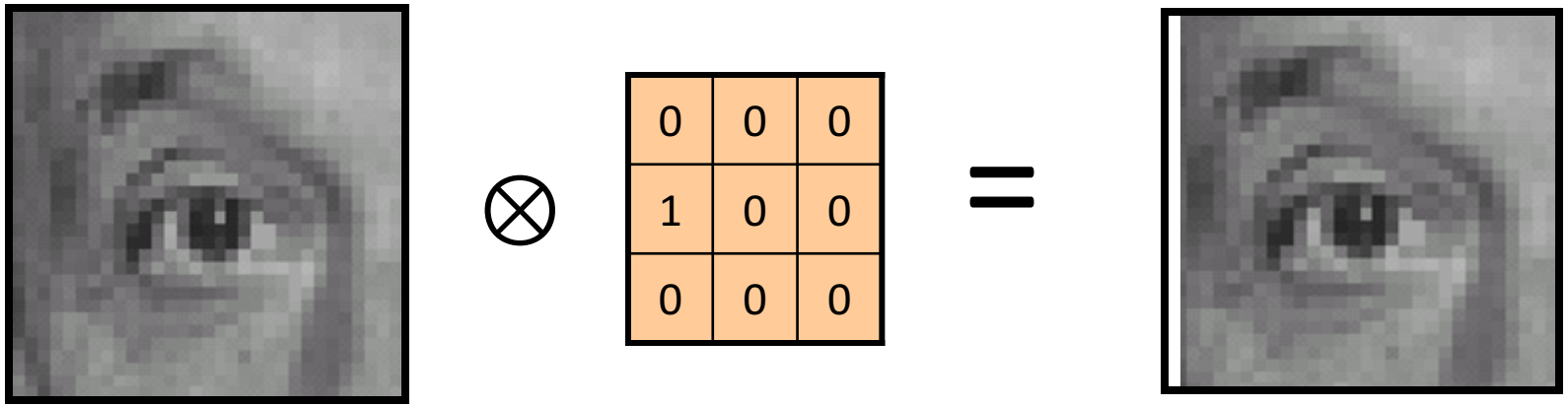


Original

Identical image

Source: D. Lowe

Linear filters: examples

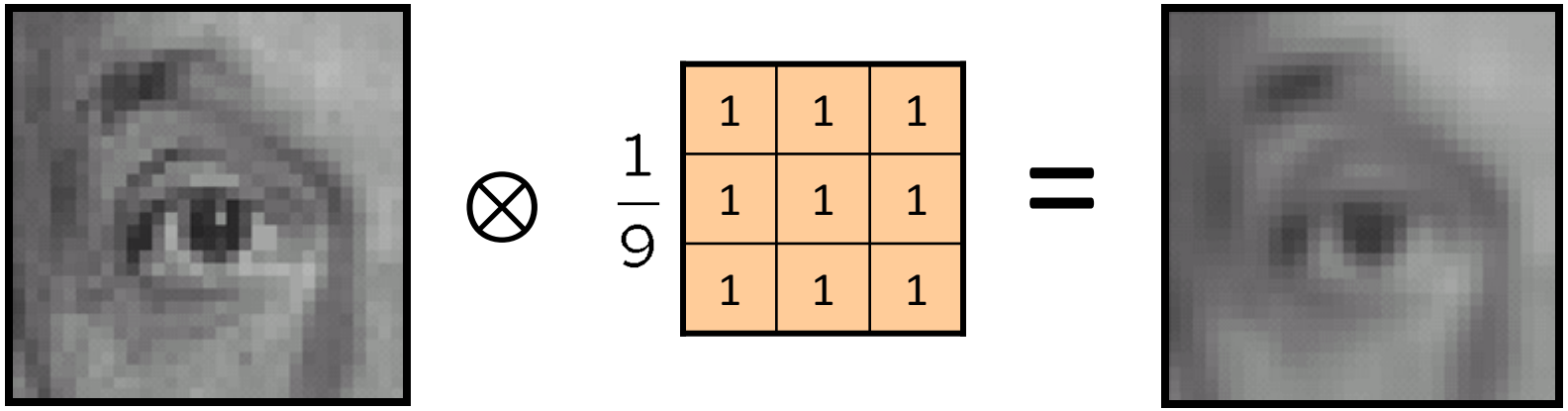


Original

Shifted right
By 1 pixel

Source: D. Lowe

Linear filters: examples

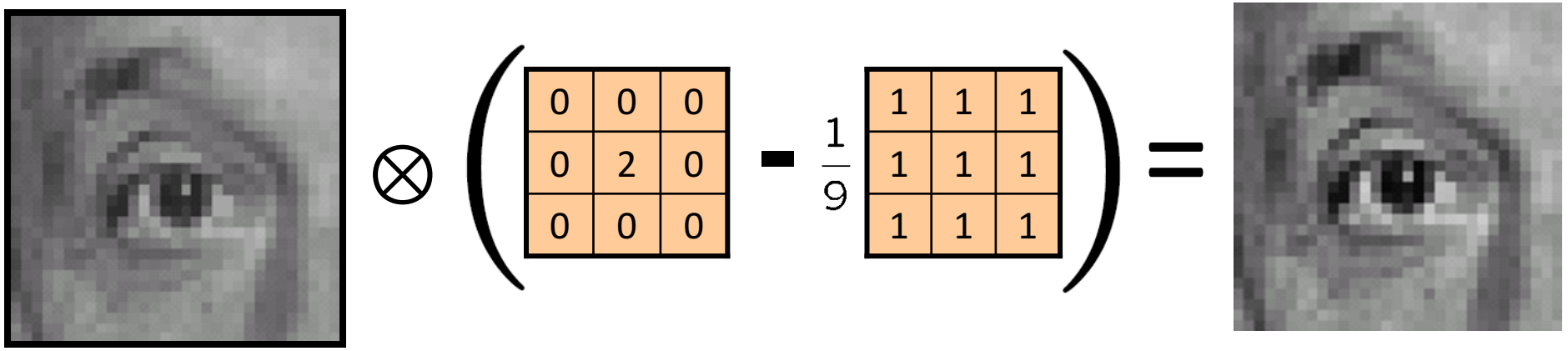


Original

Mean filter (blurring)

Source: D. Lowe

Linear filters: examples

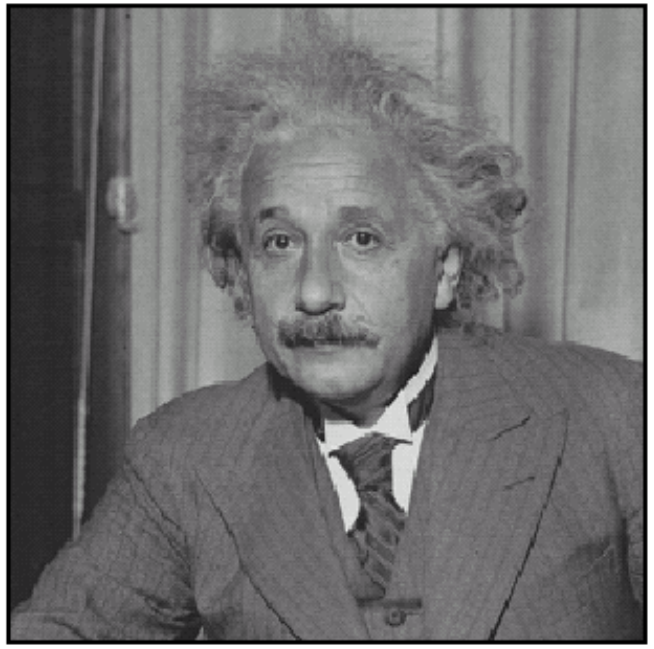


Original

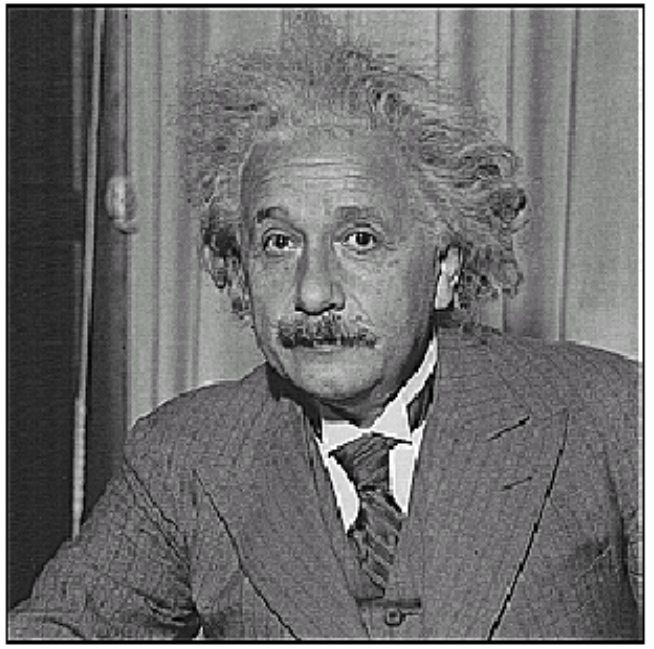
Sharpening filter
(accentuates edges)

Source: D. Lowe

Sharpening



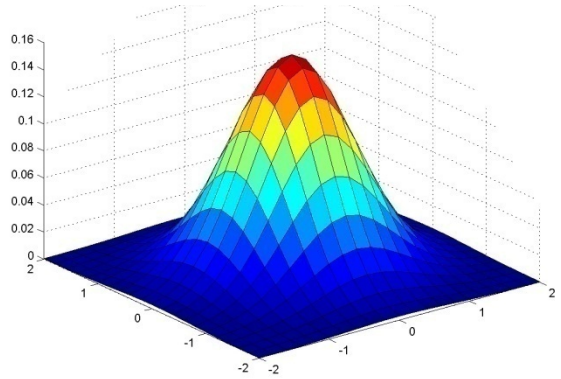
before

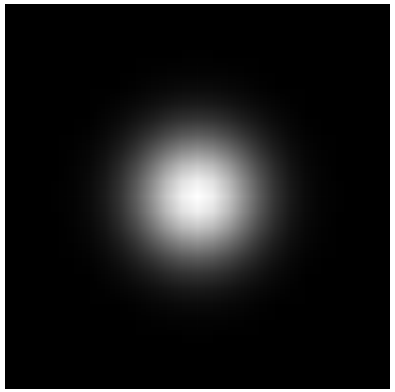


after

Source: D. Lowe

Gaussian Kernel





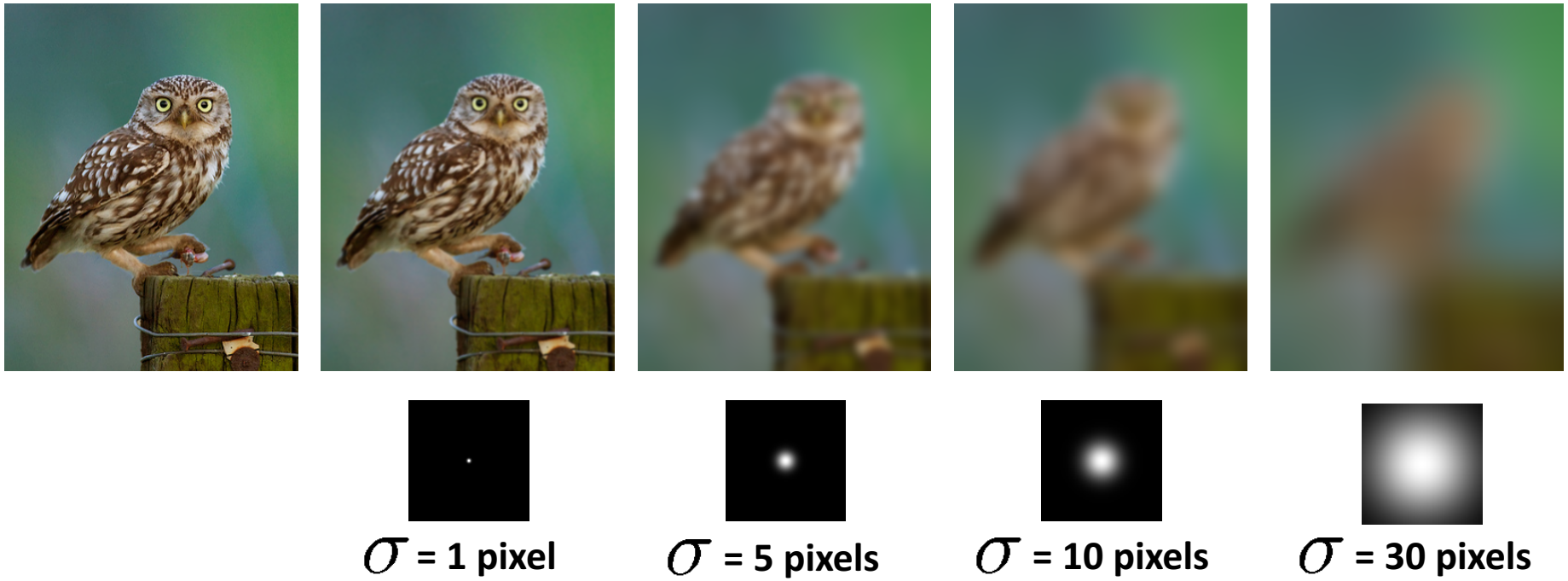
Approximated by: $\frac{1}{16}$

1	2	1
2	4	2
1	2	1

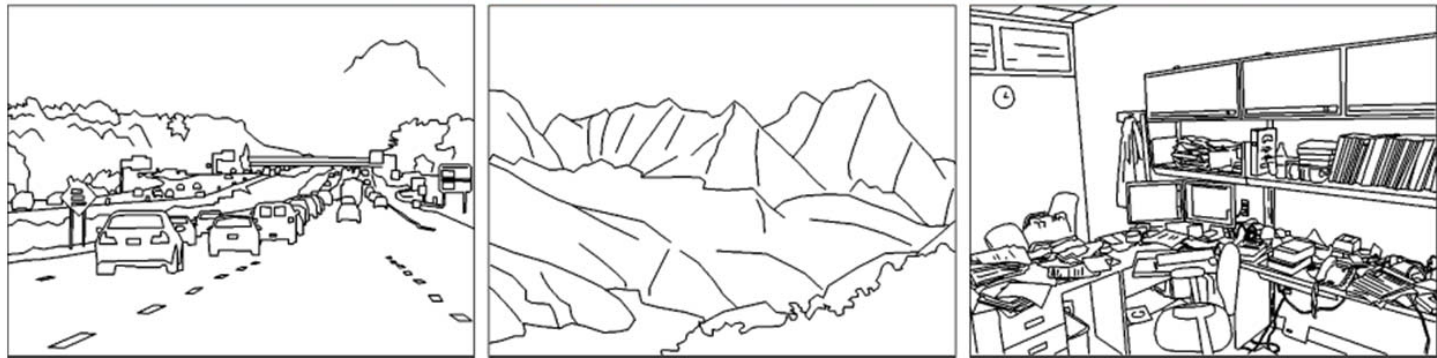
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Source: C. Rasmussen

Gaussian filter

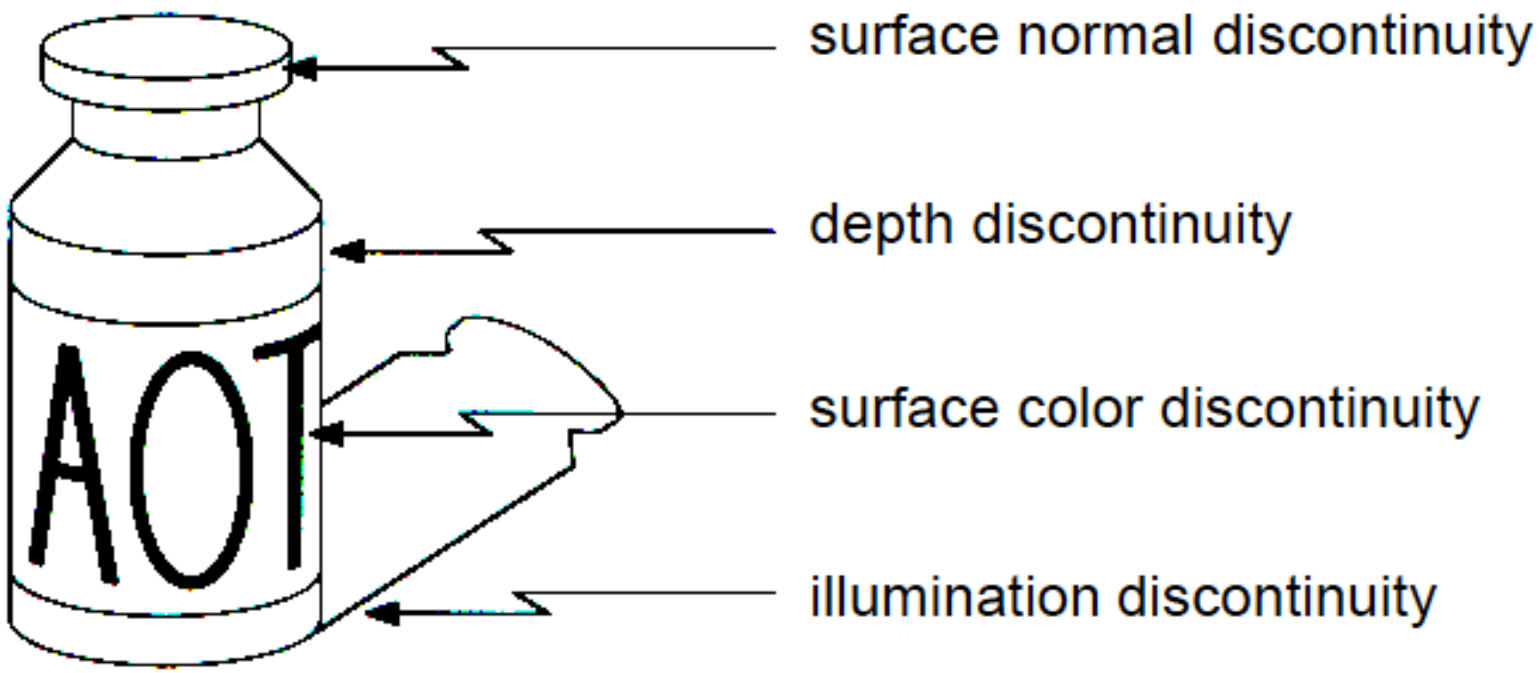


Edge Detection



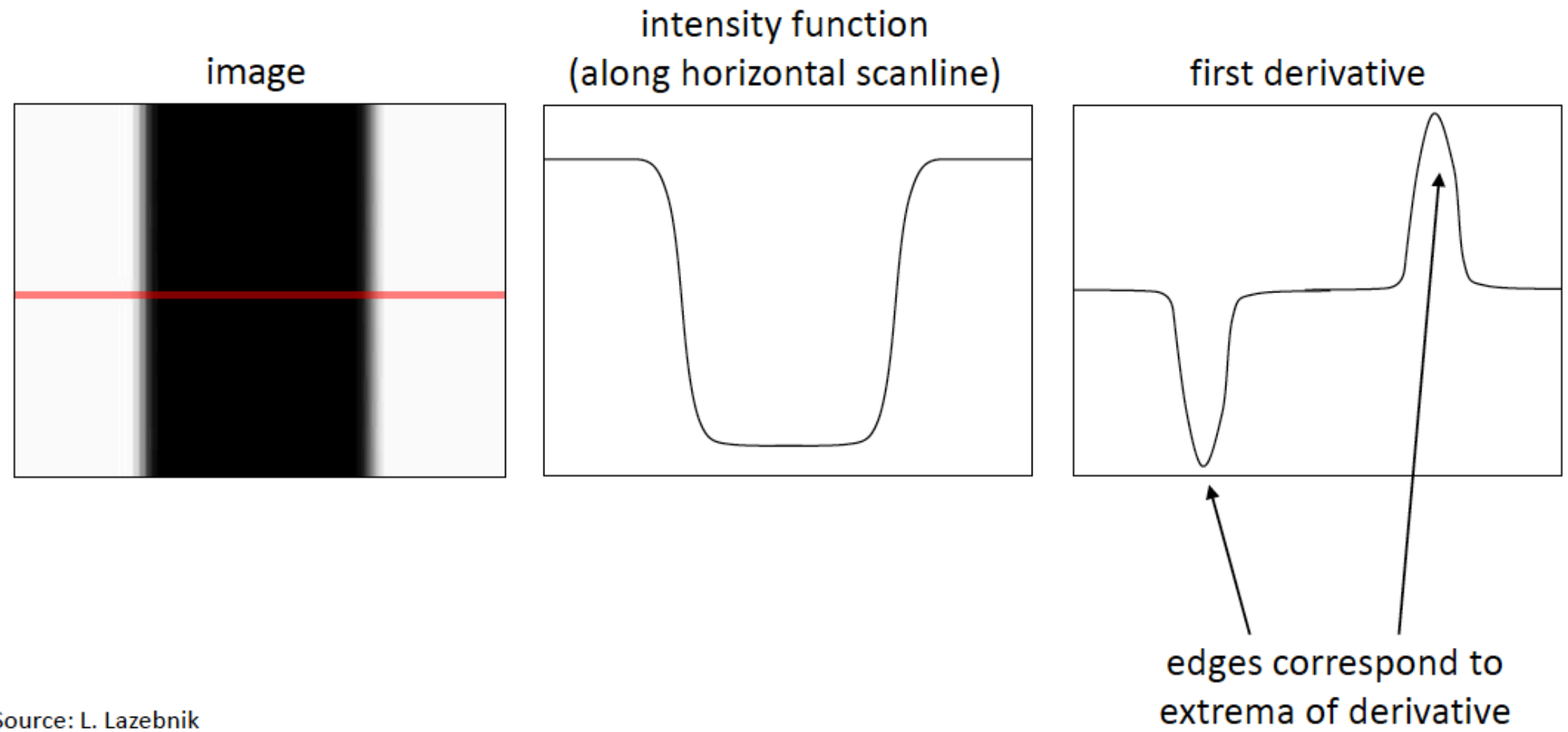
Edge Detection

Origin of Edges



Characterizing Edges

- A place of rapid change in the image intensity function



Source: L. Lazebnik

Edge Detection

Through Convolution

Sobel:

-1	0	1
-2	0	2
-1	0	1

Prewitt:

-1	0	1
-1	0	1
-1	0	1

Roberts:

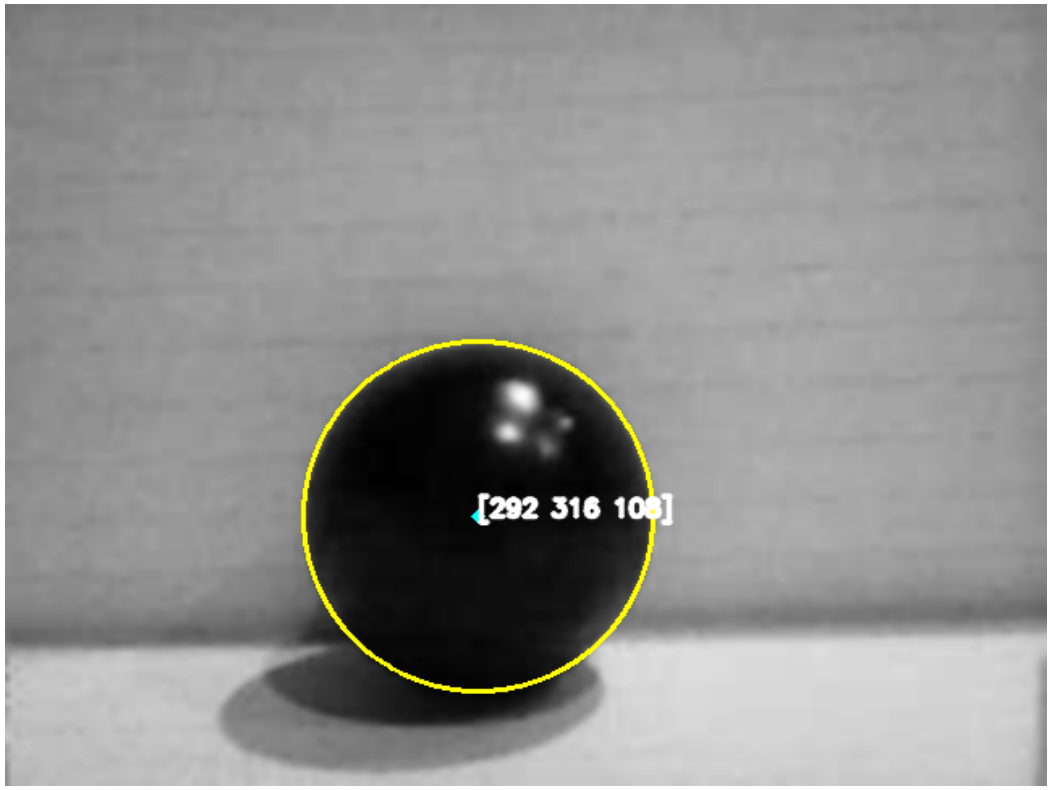
0	1
-1	0

Canny:

more complex multi-stage algorithm that uses a Gaussian filter and the intensity gradient in an image. One of the most widely used techniques.

How do we extract edges of a particular shape?

Straight Lines



Circles

Hough Transform

- Effective method for line fitting
  - Edges need not be connected
  - Complete object need not be visible
  - Key Idea: Edges **VOTE** for the possible model

Image and Parameter Spaces

Equation of Line: $y = mx + b$

Find: (m, b)

Consider point: (x_i, y_i)

$$y_i = mx_i + b \;\; or \;\; b = -x_i m + y_i$$

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

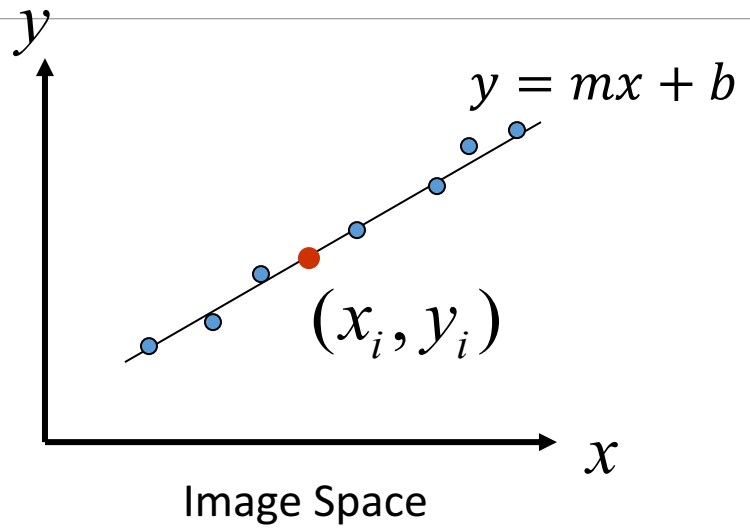
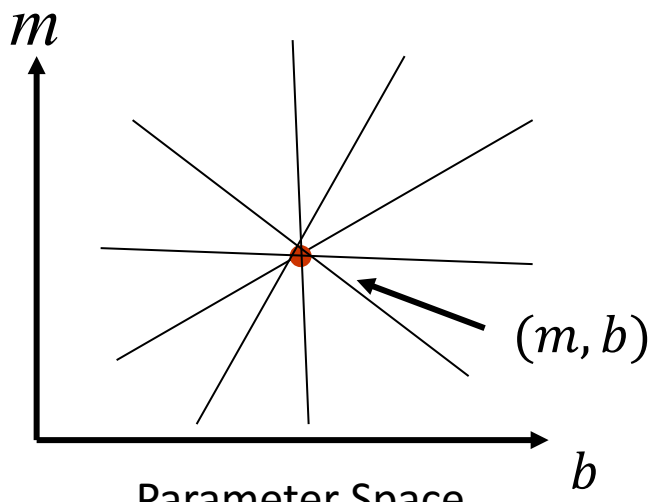


Image Space



Parameter Space

Line Detection by Hough Transform

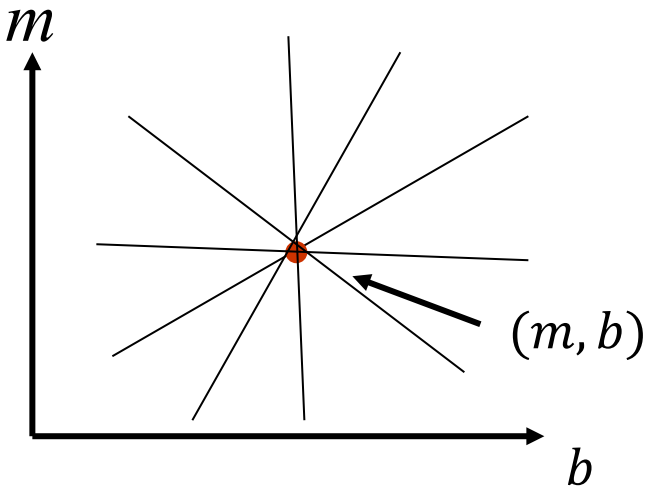
1. Create 2D accumilator array A that discretizes parameters (m, b)
2. Initialize A to zeros for all values of m and b
3. For each pixel (x_i, y_i) that is an edge:

   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

Parameter Space:



$A(m, b)$:

	1						1		
		1				1			
			1		1				
				2					
			1		1				
		1				1			
	1						1		

Single line, no noise

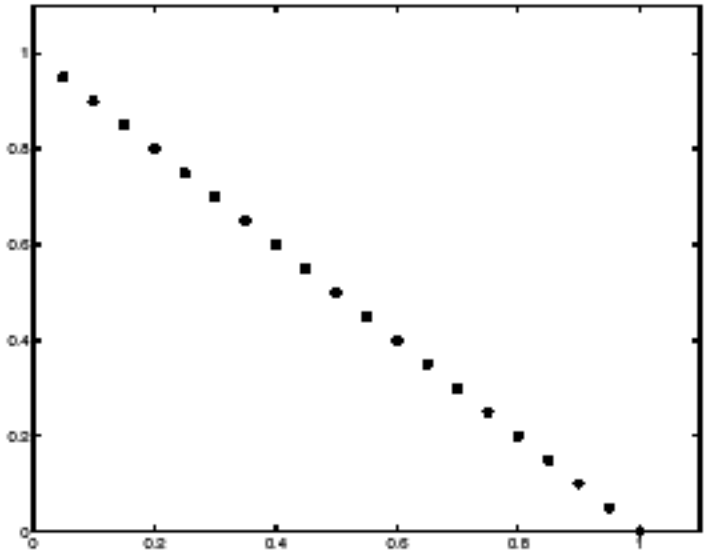
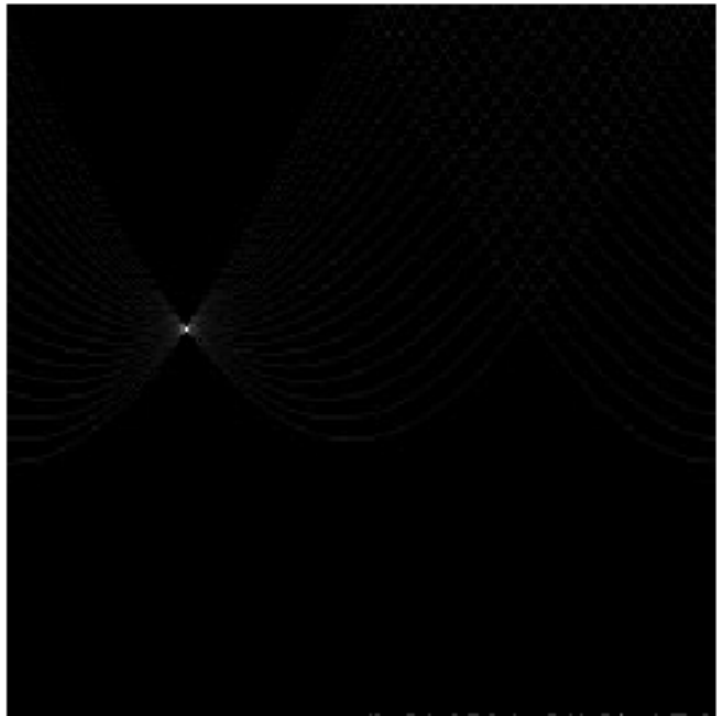


Image space



$A(m, b)$ Votes

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

Single line with noise

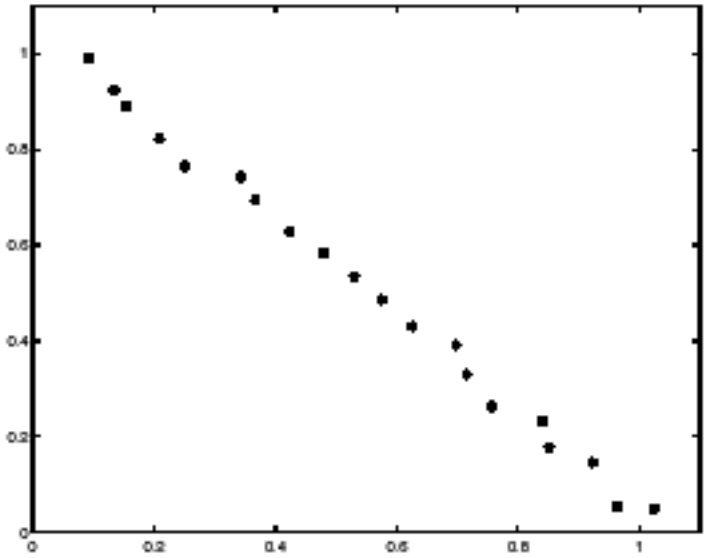
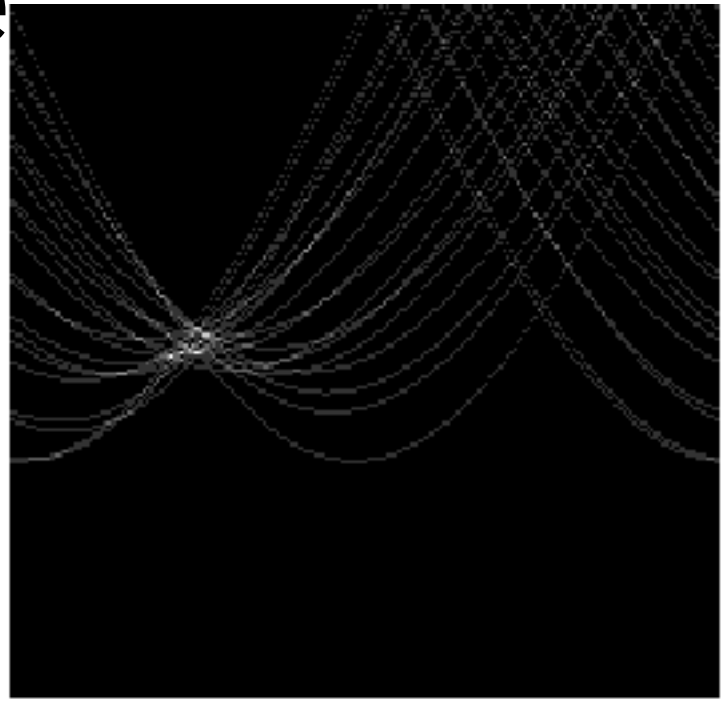


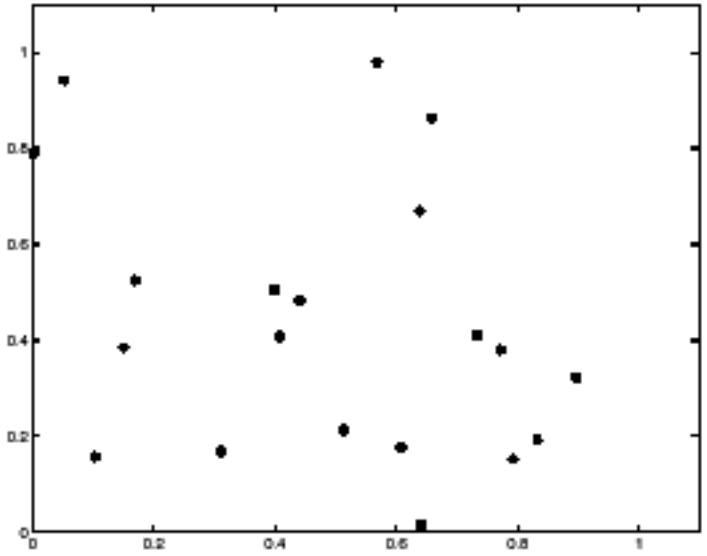
Image space

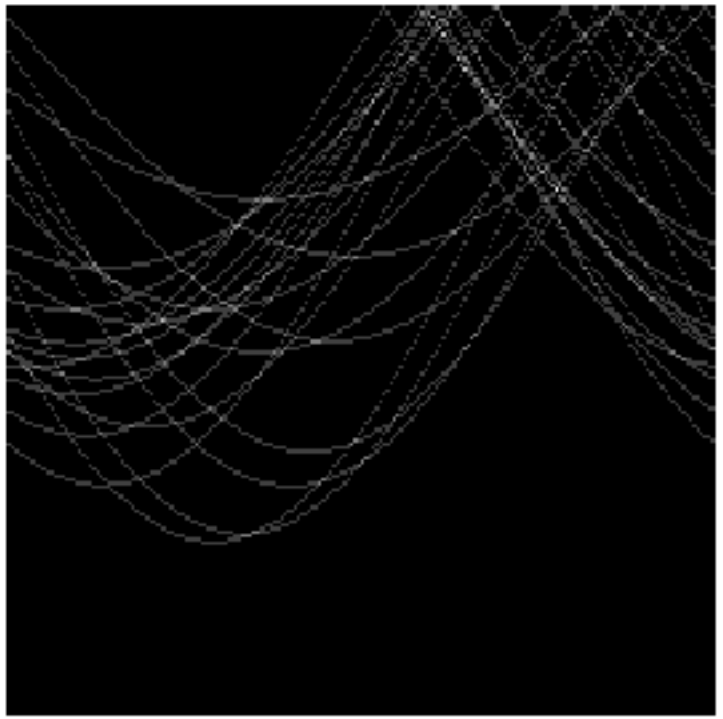


$A(m, b)$ Votes

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

No Line





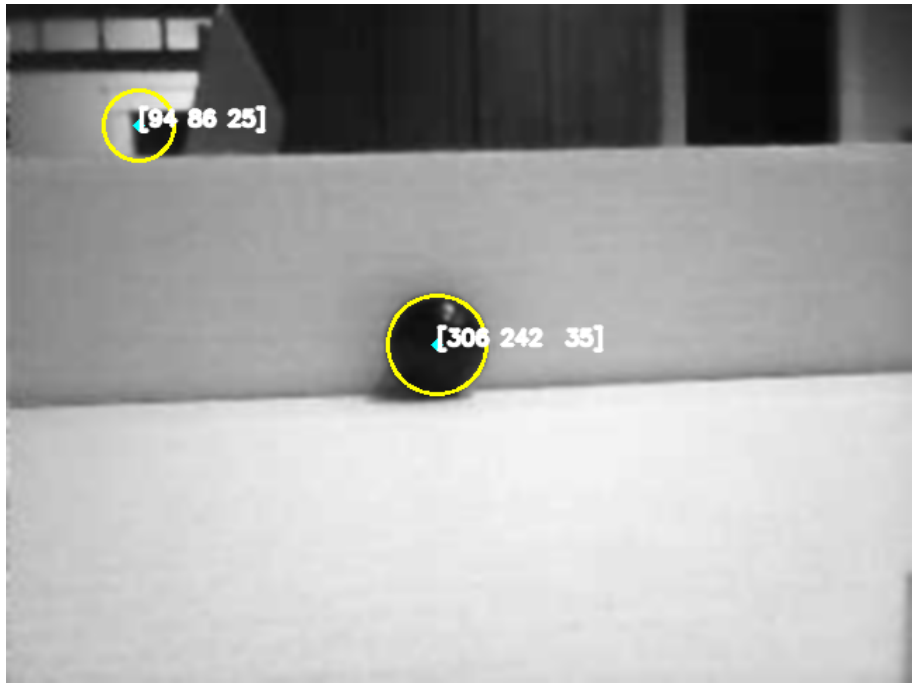
$\boldsymbol{A(m, b)}$ **Votes**

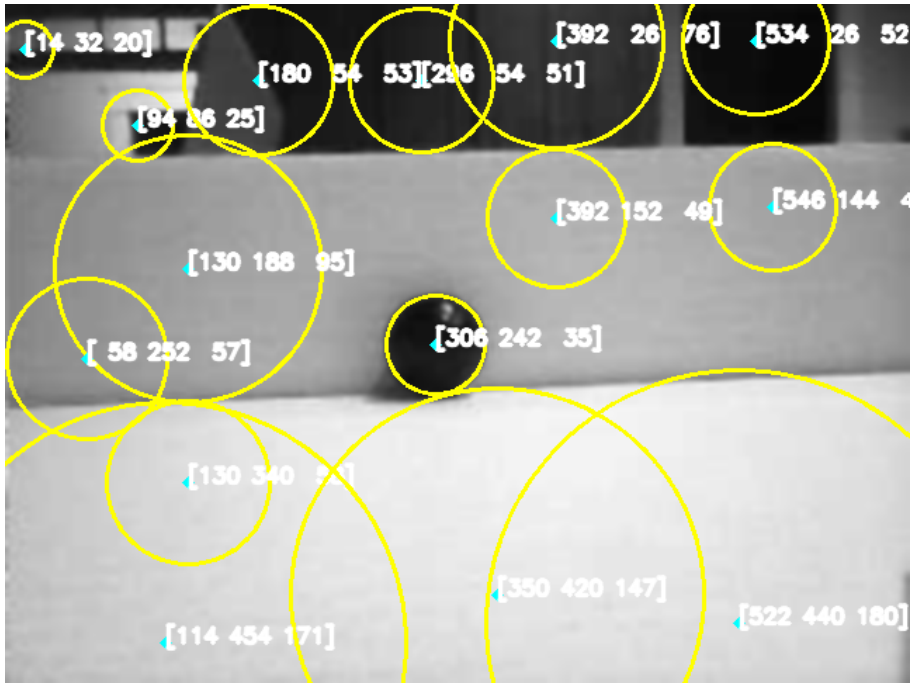
(using polar coordinates with horizontal axis as θ and vertical as rho)

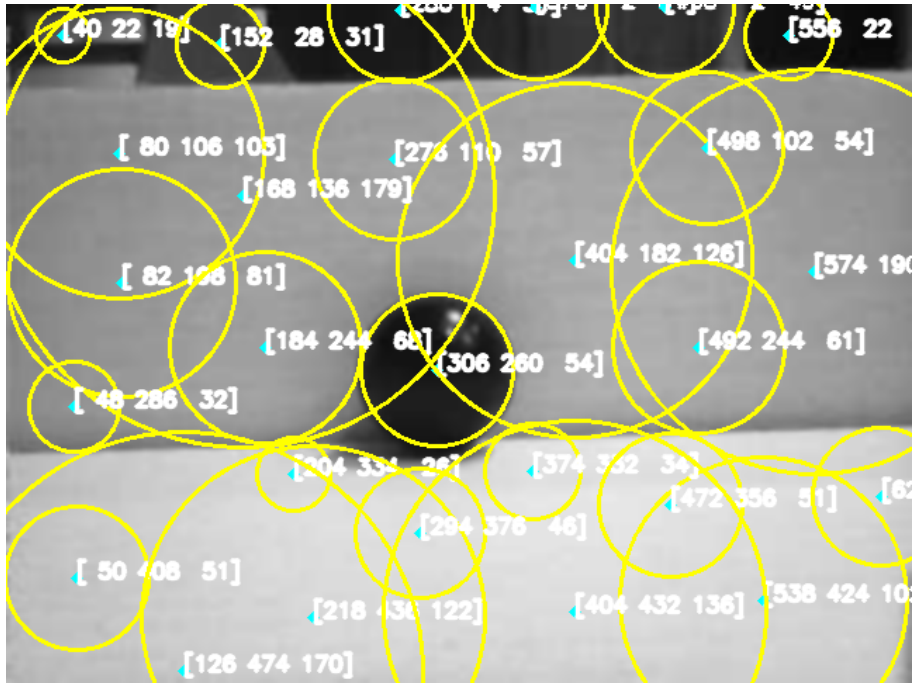
Same principle can be applied to circles
(or any other shape)

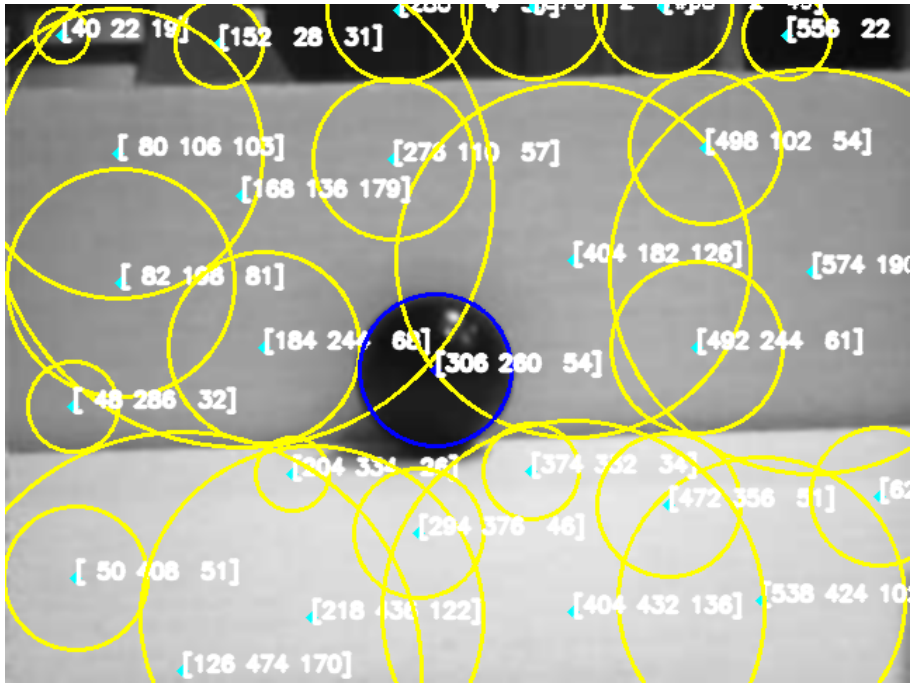
- OpenCV has a built it HoughLines() and HoughCircles() functions

- 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)


[94 86 25]
[306 242 35]


[14 32 20]
[180 54 53][296 54 51]
[94 86 25]
[392 152 49]
[546 144
[130 188 95]
[58 252 57]
[306 242 35]
[130 340
[350 420 147]
[522 440 180]
[114 454 171]


[40 22 19]
[152 28 31]
[556 22
[80 106 103]
[278 110 57]
[498 102 54]
[168 136 179]
[404 182 126]
[574 190
[82 198 81]
[184 244 68]
[306 260 54]
[492 244 61]
[48 286 32]
[374 352 34]
[472 356 51]
[294 376 46]
[50 408 51]
[218 430 122]
[404 432 136]
[538 424 10
[126 474 170]


[40 22 19]
[152 28 31]
[556 22
[80 106 103]
[278 110 57]
[498 102 54]
[168 136 179]
[404 182 126]
[574 190
[82 198 81]
[184 244 68]
[306 260 54]
[492 244 61]
[48 286 32]
[374 352 34]
[472 356 51]
[294 376 46]
[50 408 51]
[218 430 122]
[404 432 136]
[538 424 10
[126 474 170]

Hough Circles

- 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.

- 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

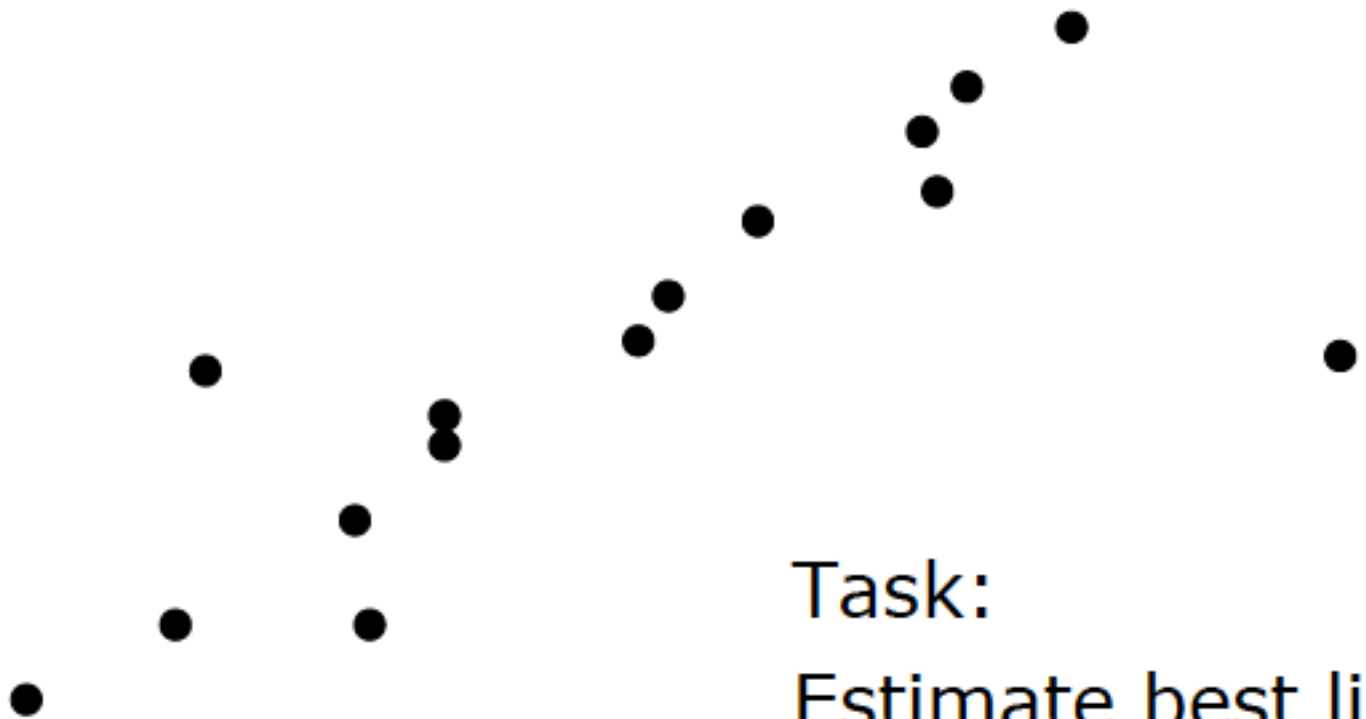
Hough transform works well for line fitting but is hard to generalize to higher dimensions

Another voting strategy: RANSAC

RANdom SAmple Concensus (RANSAC)

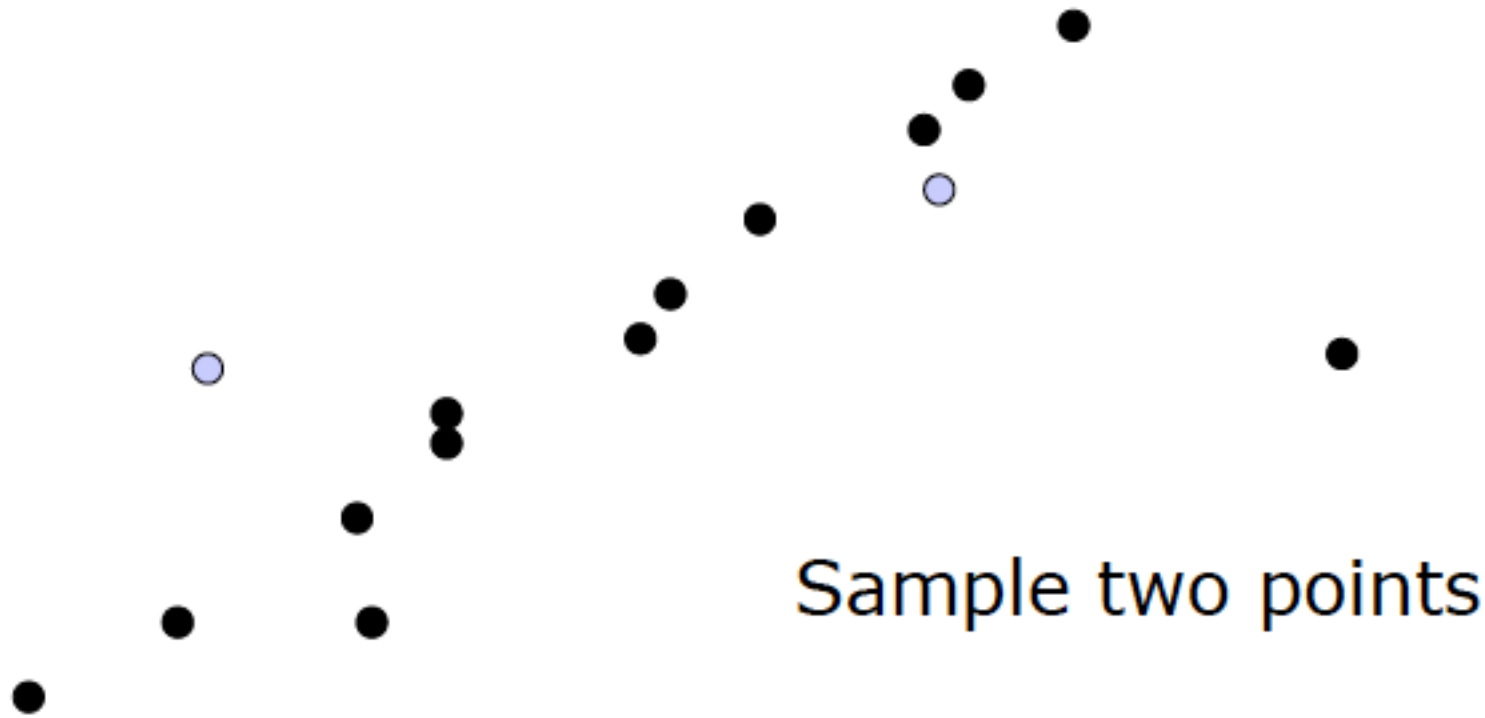
- Randomly choose s samples/points
  - Typically s = minimum sample size that lets you fit a model
- Fit a model (e.g., line) to those samples
- Count the number of inliers that approximately fit the model
- Repeat N times
- Choose the model that has the largest set of inliers

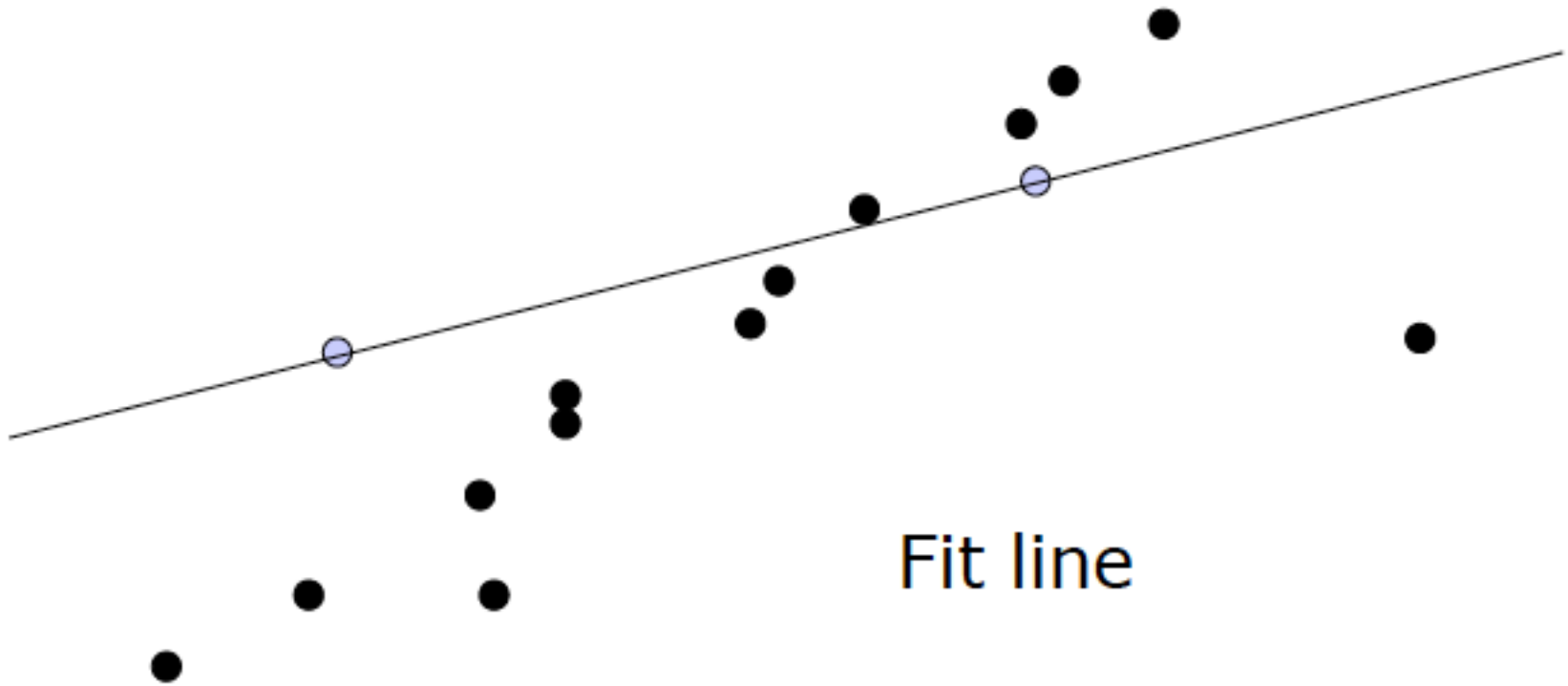
*better results are often achieved by using all the inliers from this step to create an updated model.

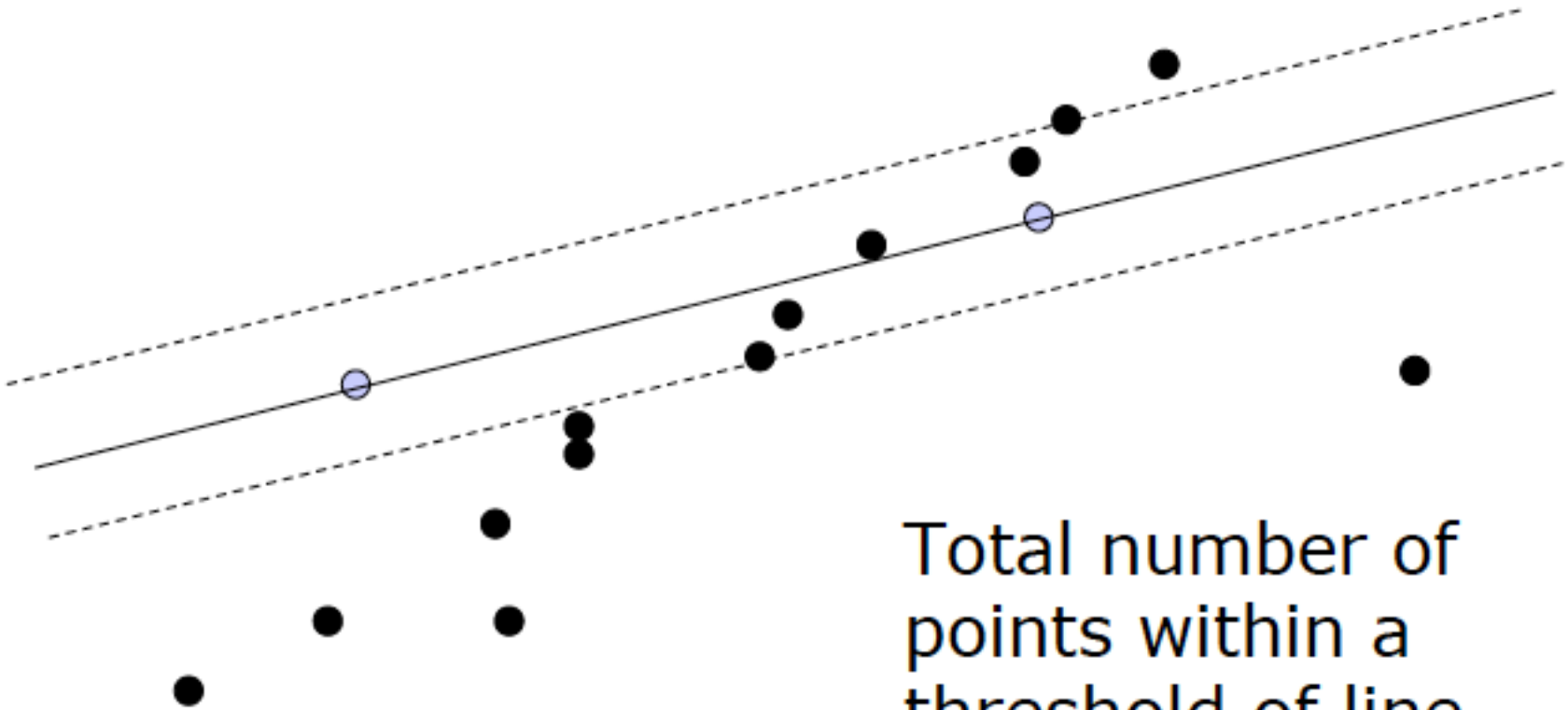


Task:

Estimate best line

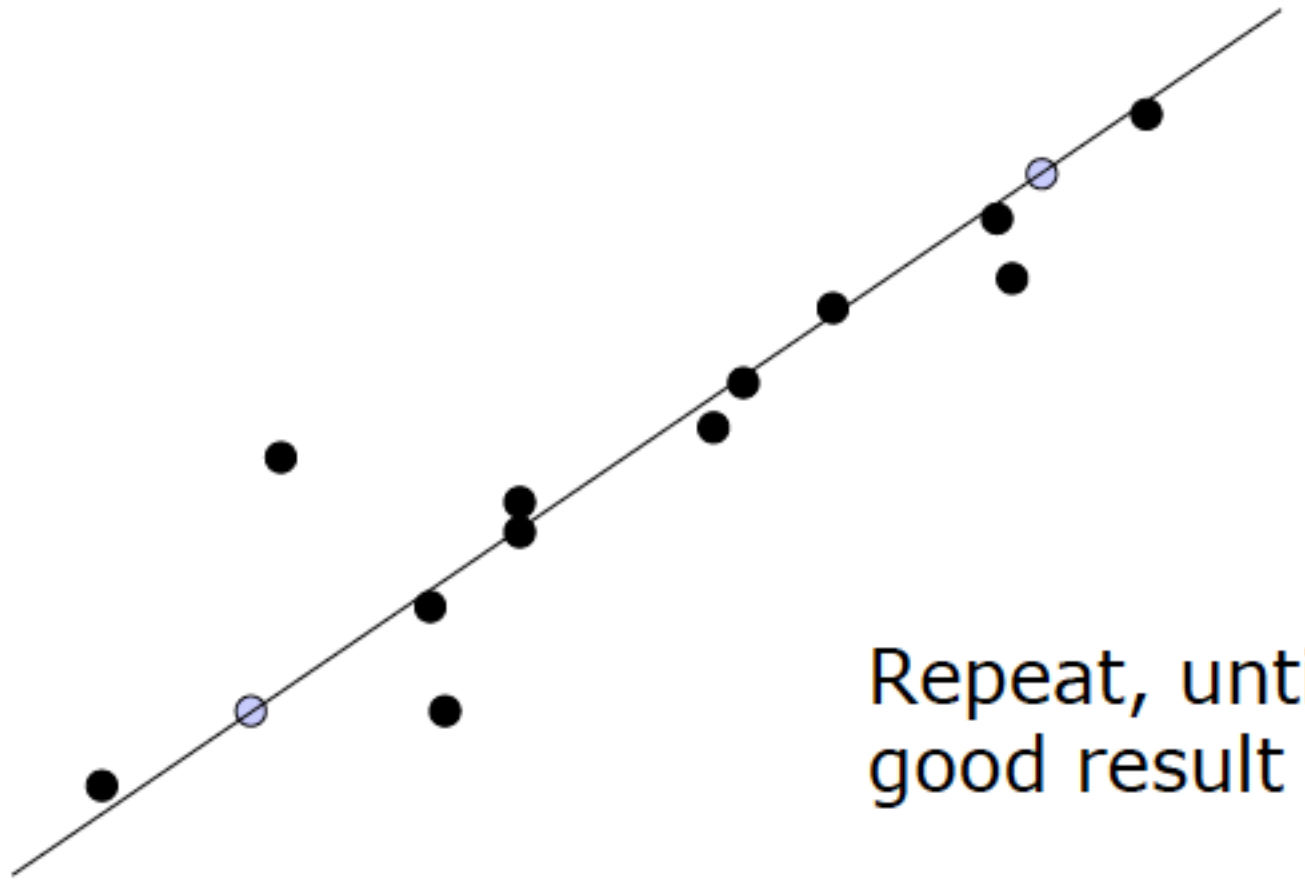

Sample two points


Fit line

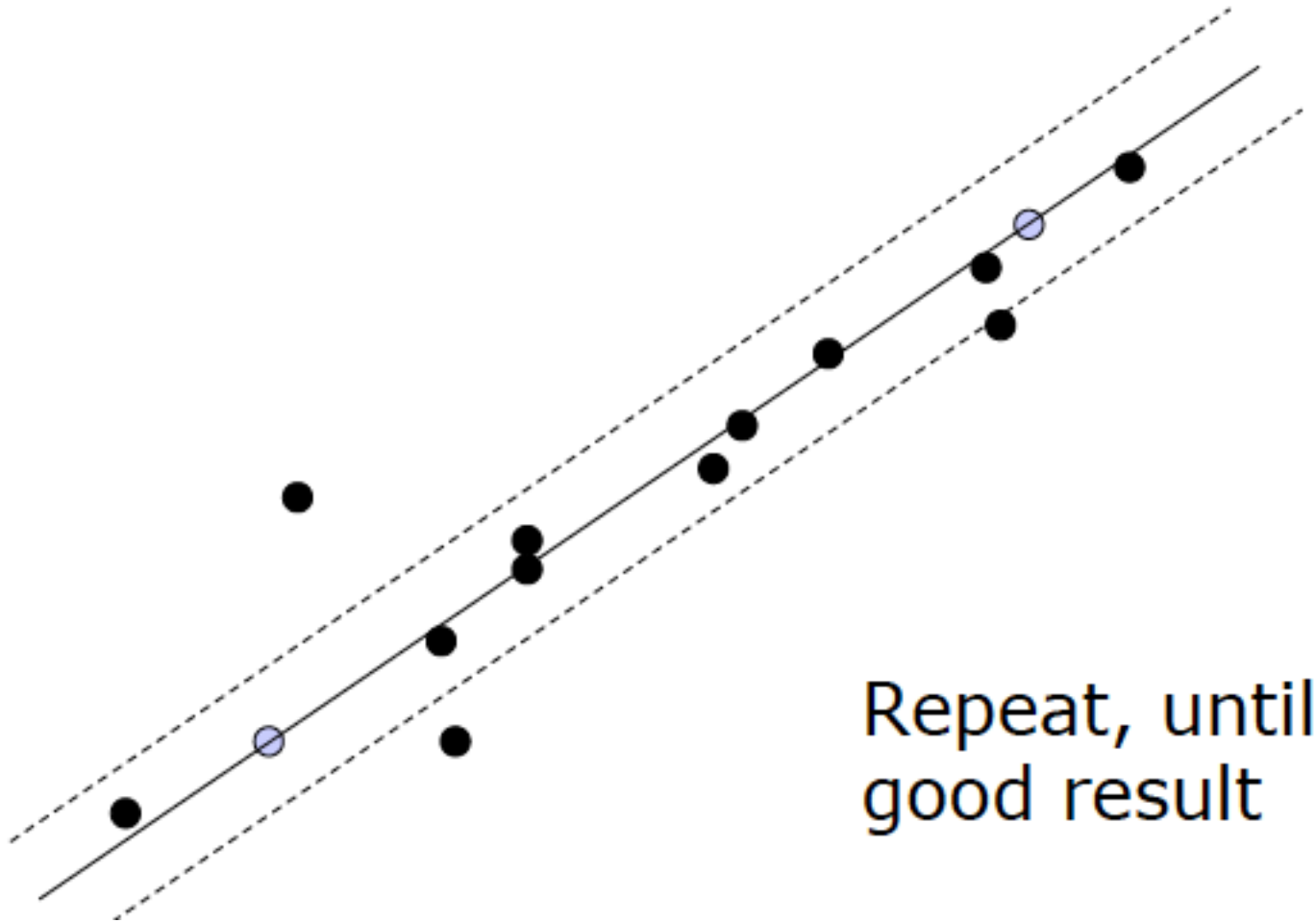


Total number of points within a threshold of line

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

Suppose 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):

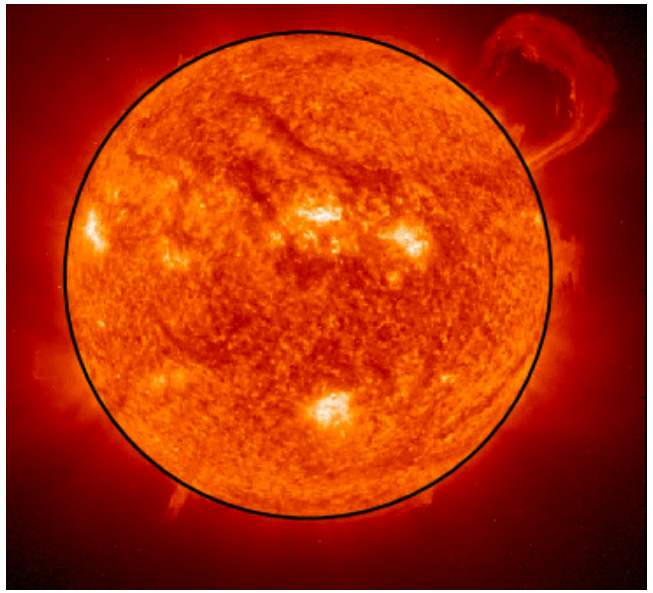
$$(1 - w^n)^N = 1 - p$$

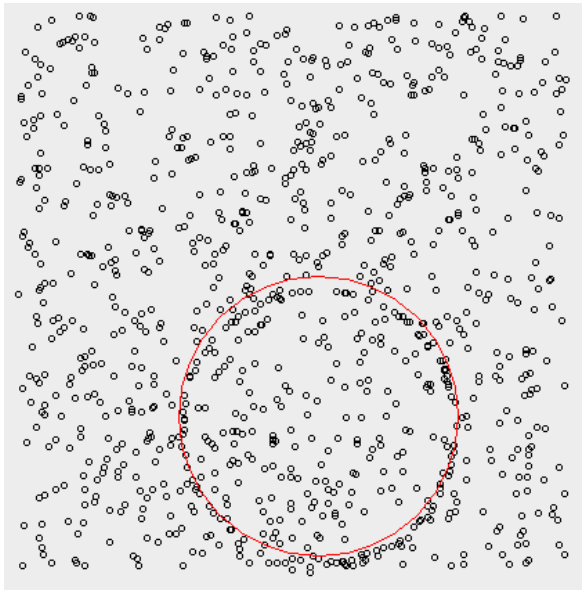
$$N = \frac{\log(1 - p)}{\log(1 - w^n)}$$

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

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.

- We've tried to keep the **lighting conditions** similar across all images for this lab, but when running the robot in the future you might find yourself in a place with significantly different lighting. Normalizing the image can help reduce lighting effects.

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

References

- Image processing fundamentals
  - Autonomous Mobile Robots 4.3
  - Robotics, Vision and Control 12.2-12.4
- Hough transforms
  - Autonomous Mobile Robots page 205
  - Robotics, Vision and Control 13.2
- RANSAC
  - Autonomous Mobile Robots 4.7.2.4
  - Robotics, Vision and Control 14.2.3

Quiz 1

- Quiz 1 will be during the last 30 minutes of class on Thursday
- Material covered: today’s lecture
- Practice problems available on Piazza under Resources
- I will take additional questions about the material on Thursday