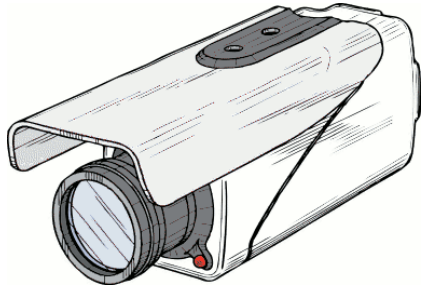
Univerza v Ljubljani

ViCOS Visual Cognitive Systems Lab

Machine perception
Derivatives and edge detection



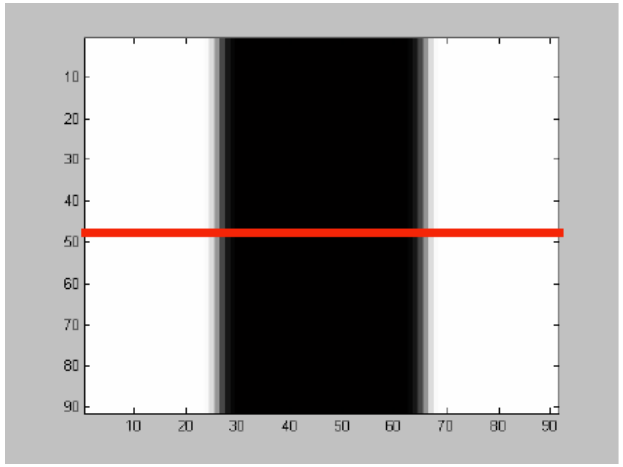
Matej Kristan

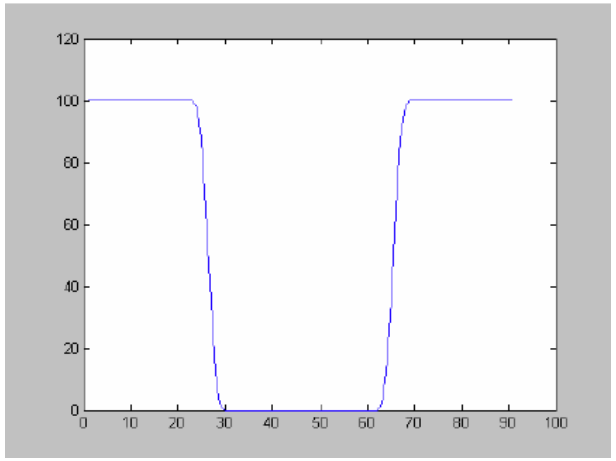
Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

Machine perception

IMAGE DERIVATIVES

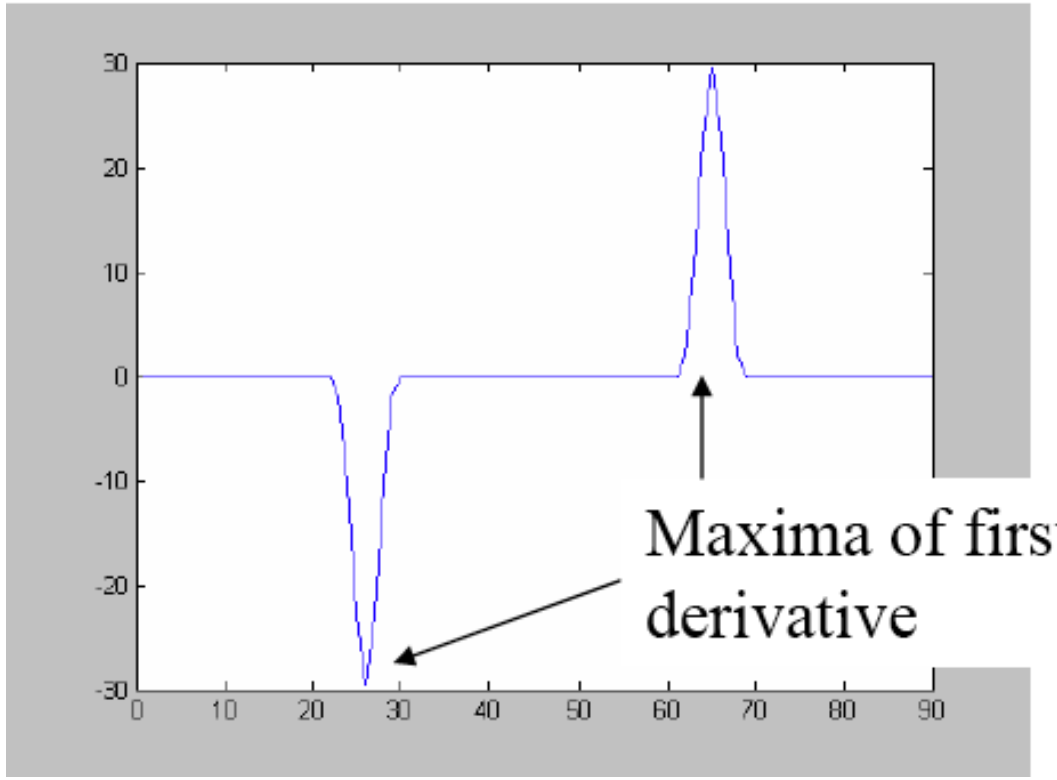
1D derivative: Intuition





First order derivative

$$\frac{d}{dx} f(x)$$



Derivatives and convolution

- A partial derivative of a continuous 2D function $f(x,y)$:

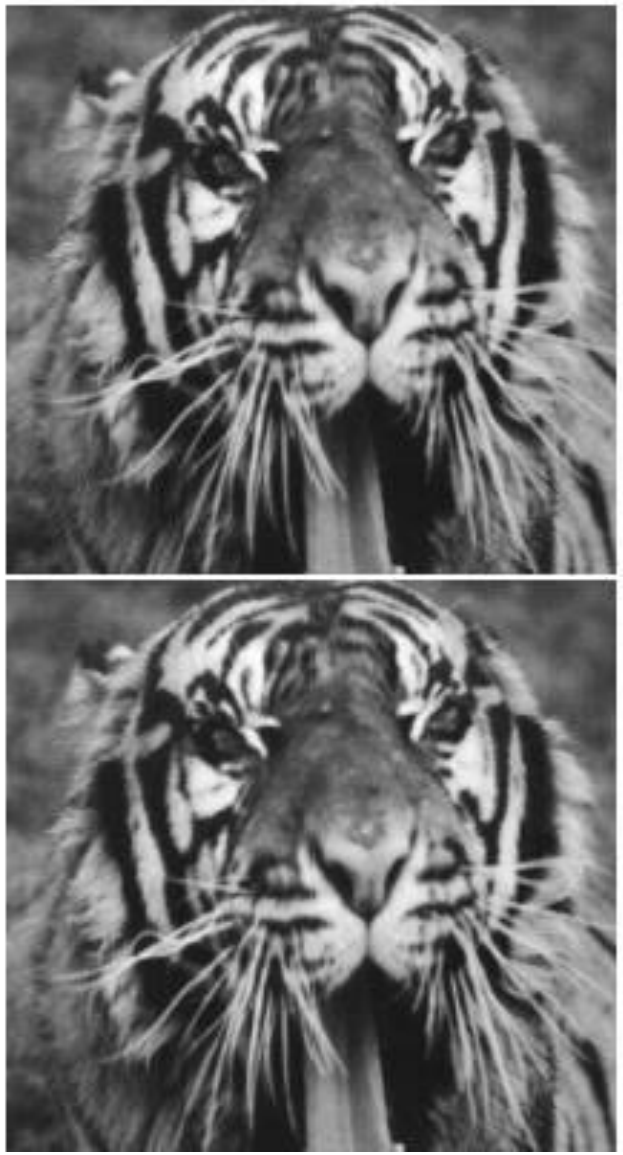
$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

- For discrete case, approximate by using finite differences:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

- Question: If implemented by convolution, what would the convolution kernel for derivative look like? (Next slide)

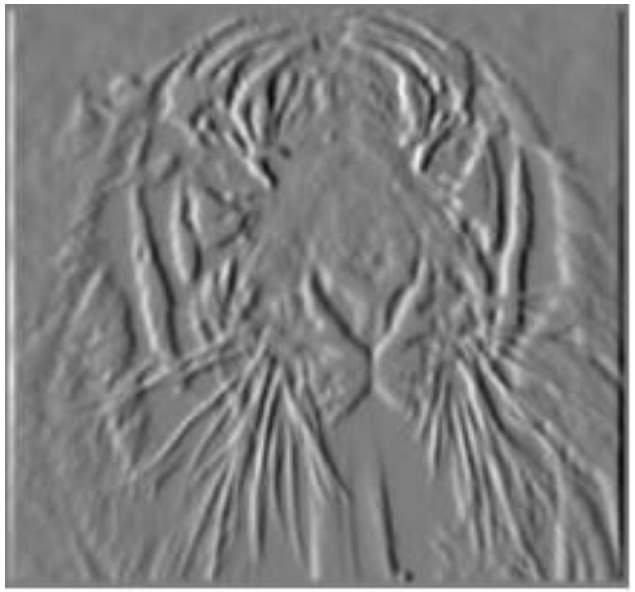
Partial derivatives: Implementation



Horizontal derivative

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y)-f(x,y)}{1}$$

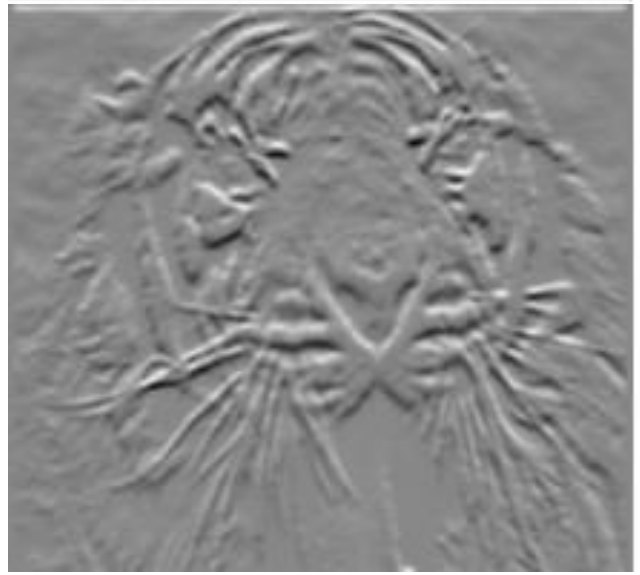
* | -1 | 1 | =



Vertical derivative

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+1)-f(x,y)}{1}$$

* | -1 / 1 | =

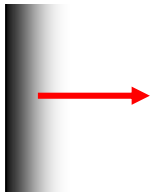


Partial derivatives: Image gradient

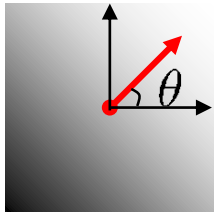
- Image gradient: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

- Gradient points in direction of greatest intensity change:

$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$



$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$



$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

- Gradient direction (orientation of edge normal):

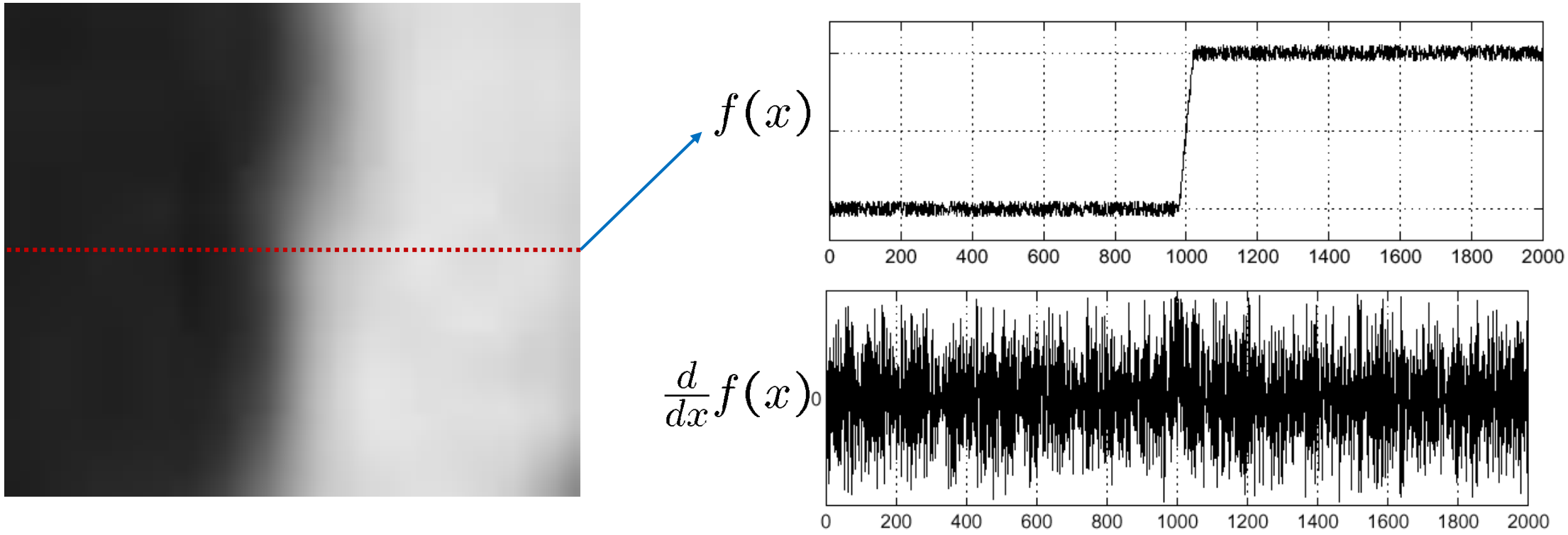
$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- Gradient strength is defined by its magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Discrete world is noisy...

- Take a single line in the image:
  - Plot intensities w.r.t. pixels:



So where did the edge go!? Noise gets amplified by derivation...

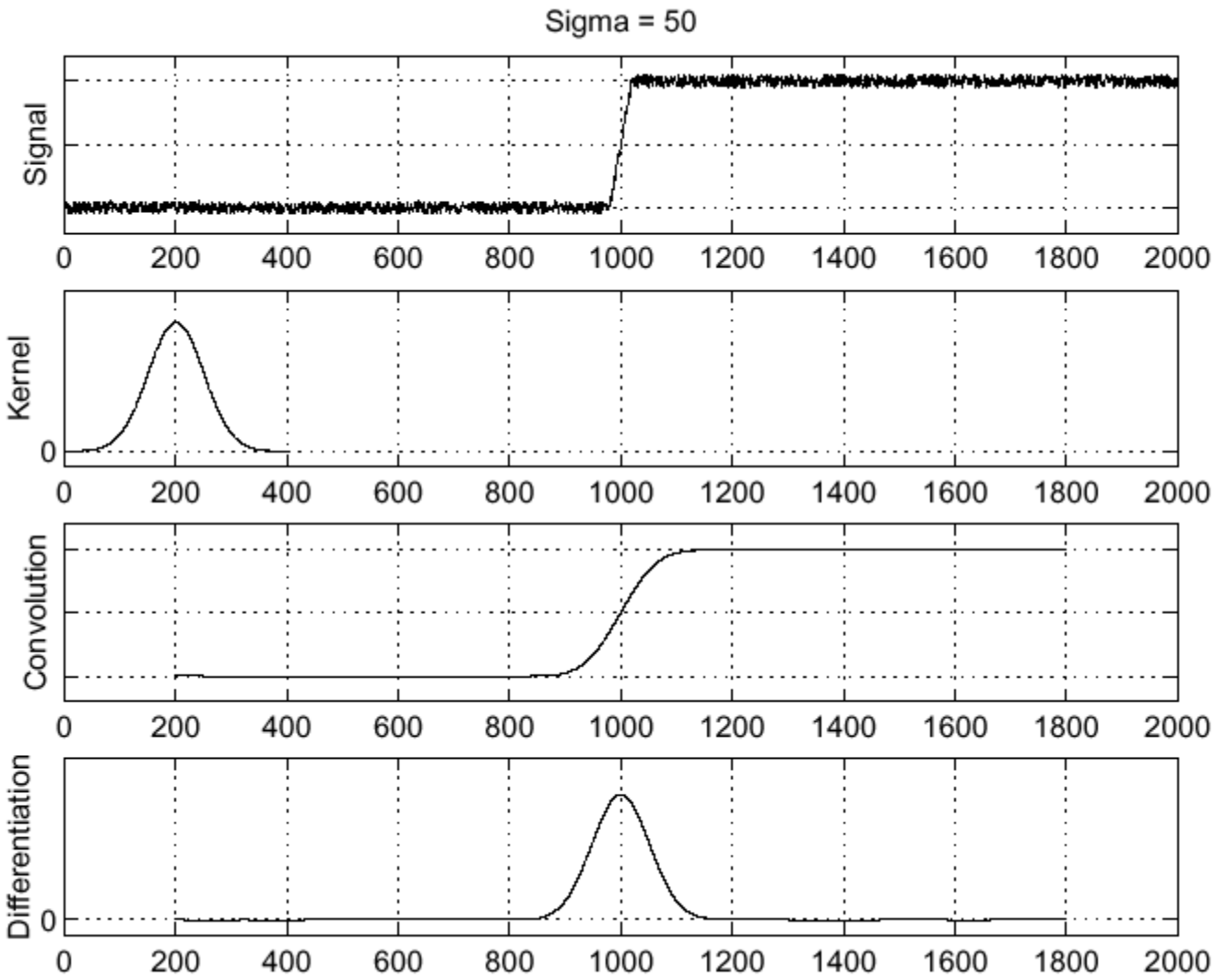
Solution: Smooth the image first

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



Where's the edge?

Find maxima in map $\frac{\partial}{\partial x}(h \star f)$

Slide credit: Steve Seitz

Remember convolution properties

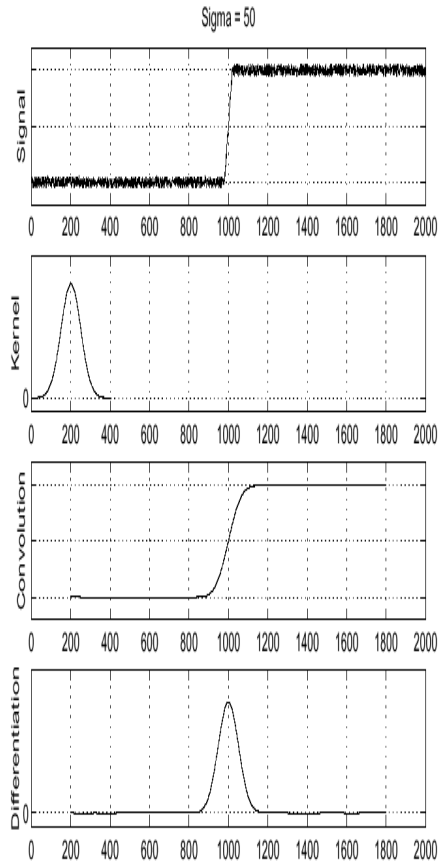
- Derivatives: $\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$

f

h

$h \star f$

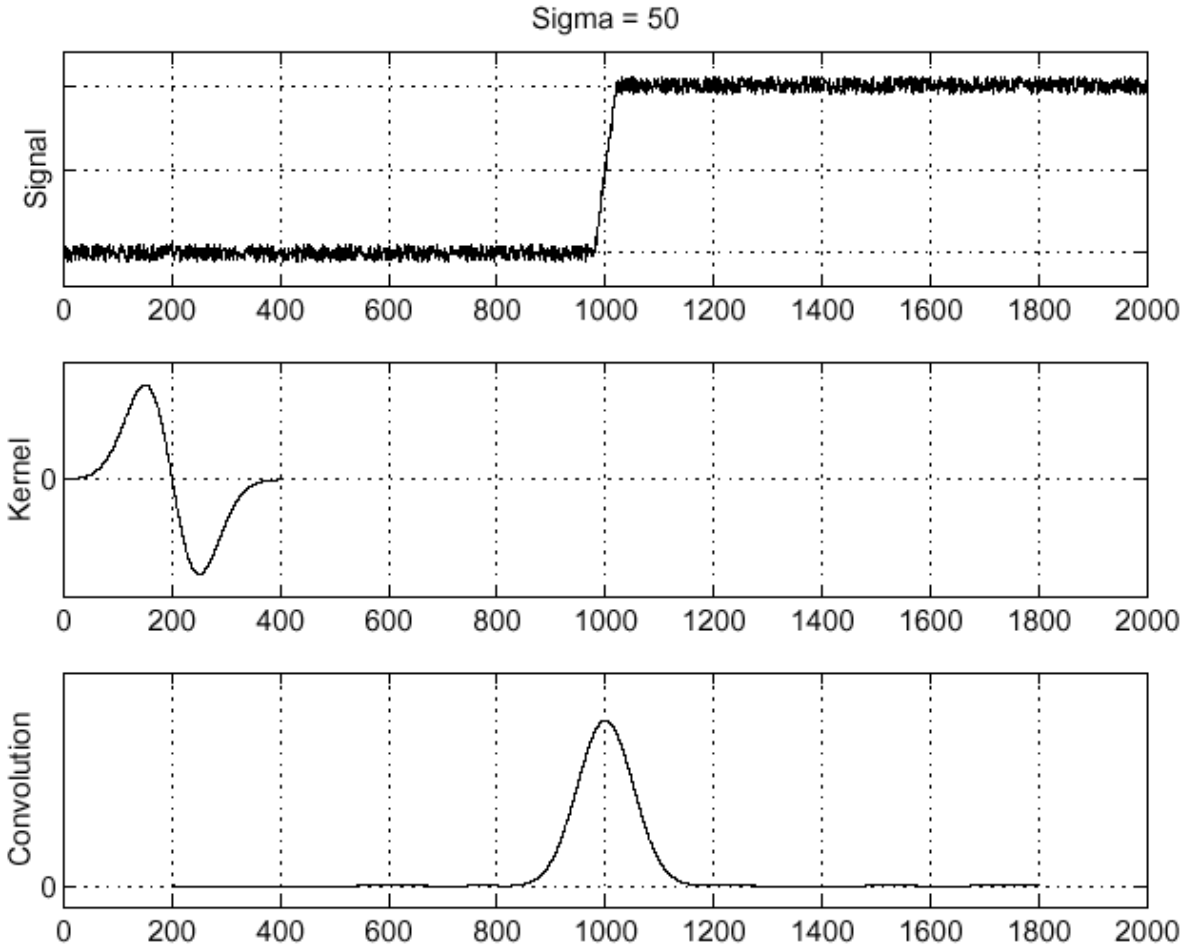
$\frac{\partial}{\partial x}(h \star f)$



f

$\frac{\partial}{\partial x}h$

$(\frac{\partial}{\partial x}h) \star f$



2D partial derivatives – naiive approach

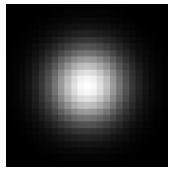
1. Smooth the image by a 2D Gaussian filter
2. Take derivative w.r.t. x

1 Blurring

Image

*

Gaussian kernel



=

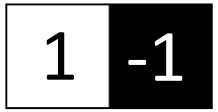
Blurred image

2 Differentiating w.r.t. x

Blurred image

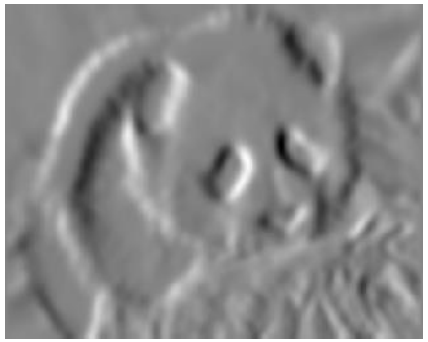
*

Derivative kernel



=

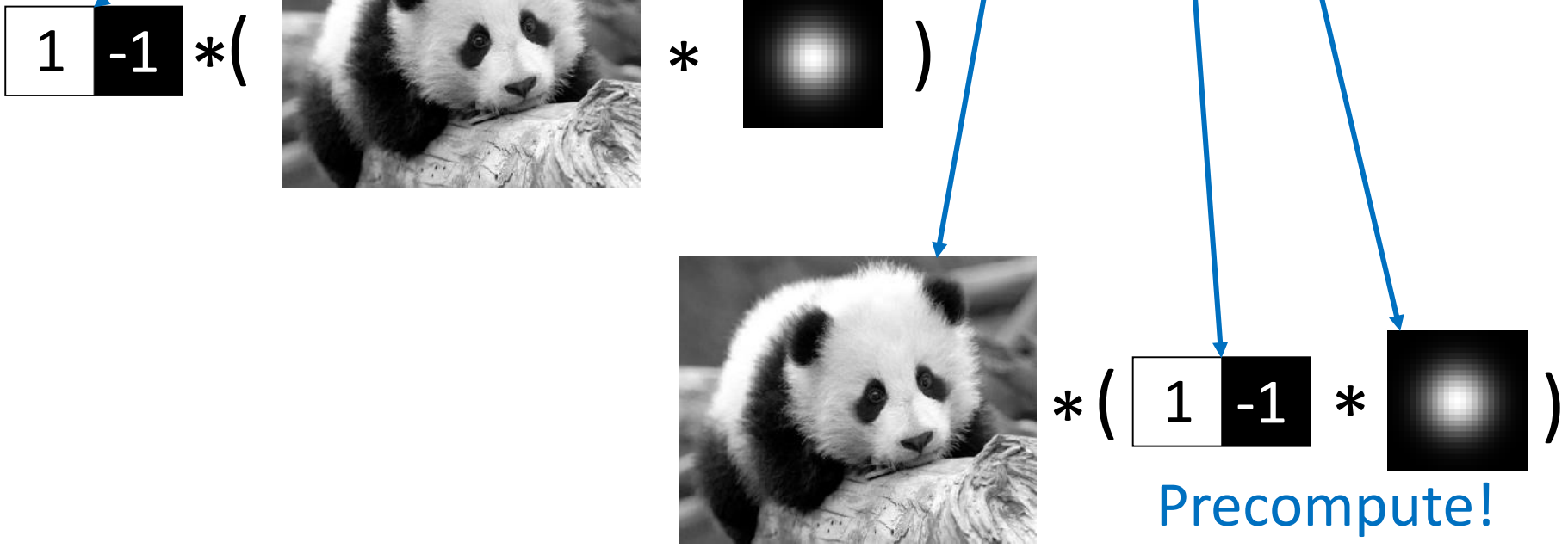
Derivative image



2D partial derivatives – smarter approach

- Recall the convolution property:

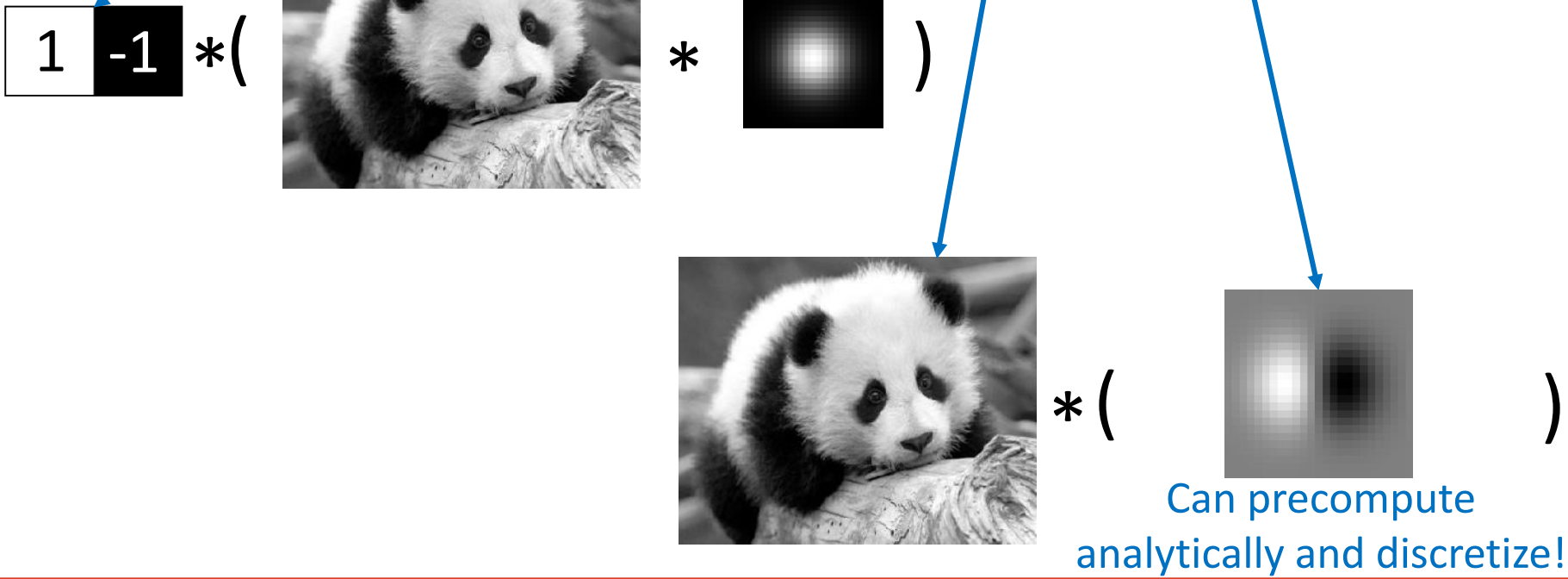
$$\frac{\partial}{\partial x}(I * G) = I * (\frac{\partial}{\partial x} G)$$



Smarter way

- Recall the convolution property:

$$\frac{\partial}{\partial x}(I * G) = I * (\frac{\partial}{\partial x} G)$$

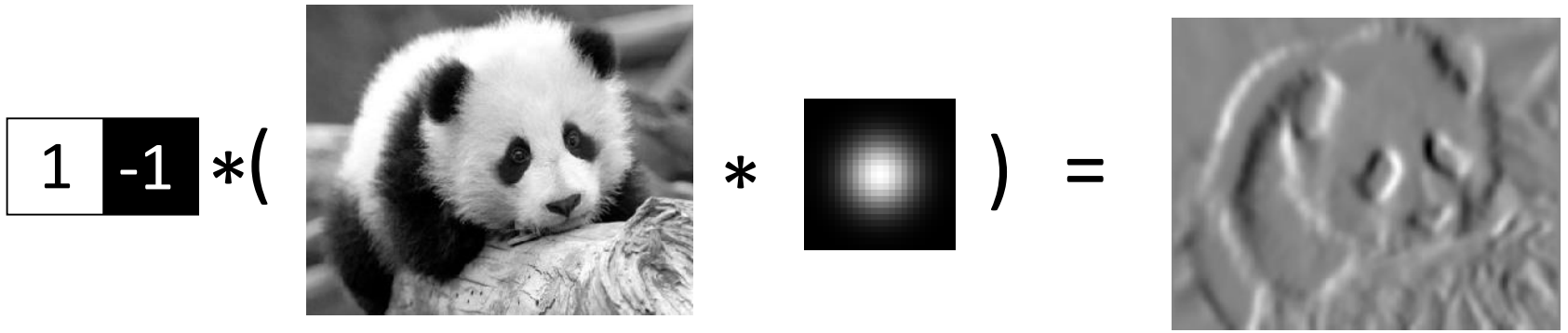


2D partial derivatives – smarter approach

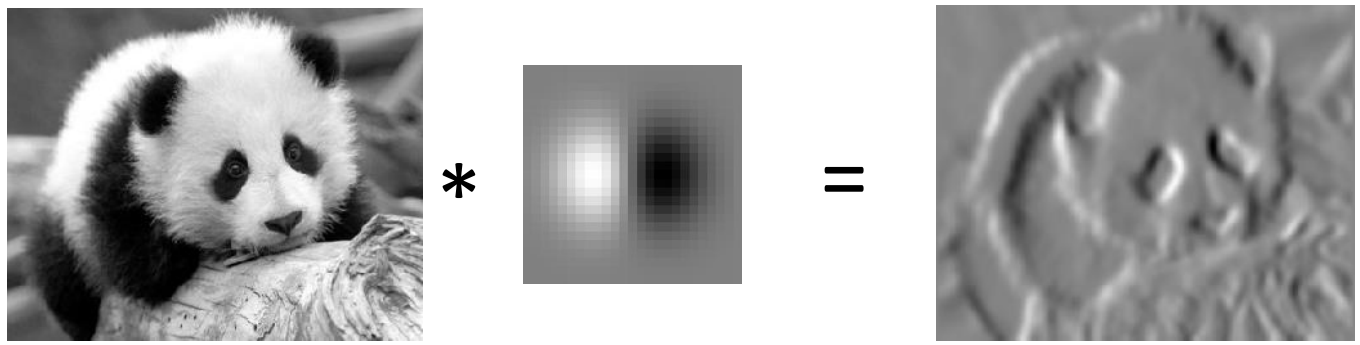
- Recall the convolution property:

$$\frac{\partial}{\partial x}(I * G) = I * (\frac{\partial}{\partial x} G)$$

Naiive:

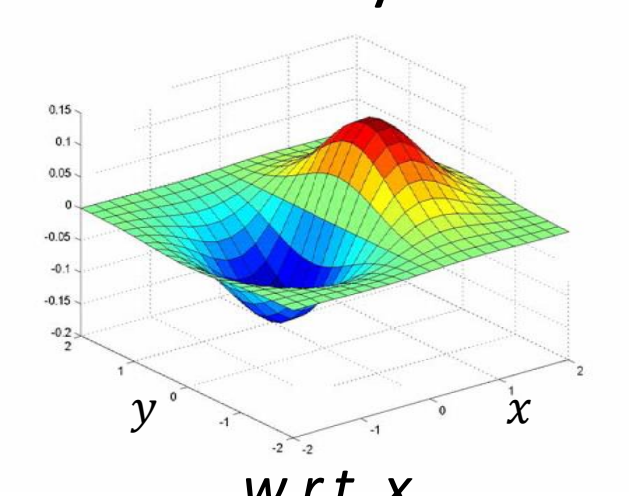


Smarter:

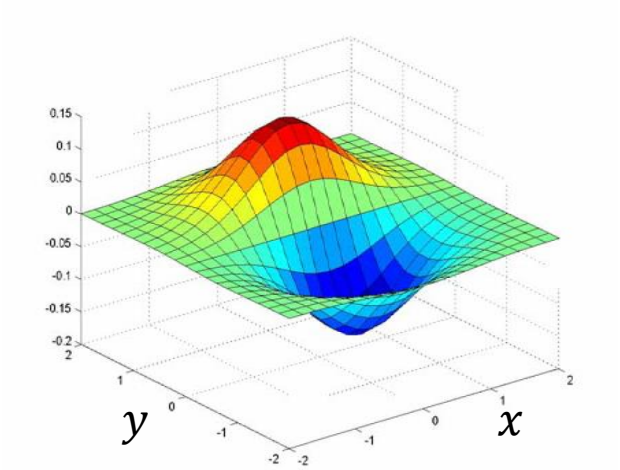


Gaussian partial derivatives

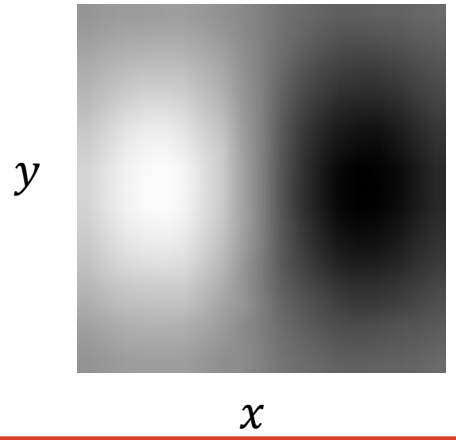
- Convolution kernels for taking partial derivatives w.r.t. x and y:

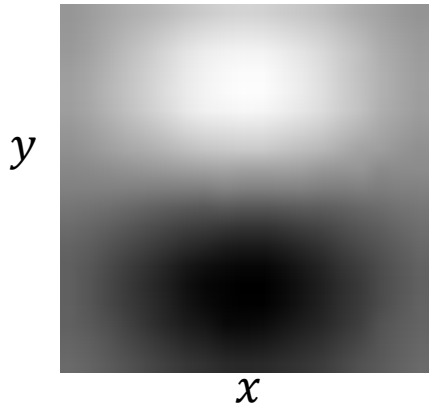


w.r.t. x



w.r.t. y





Some other popular kernels

Prewitt: $M_x =$

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

; $M_y =$

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

Sobel: $M_x =$

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

; $M_y =$

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

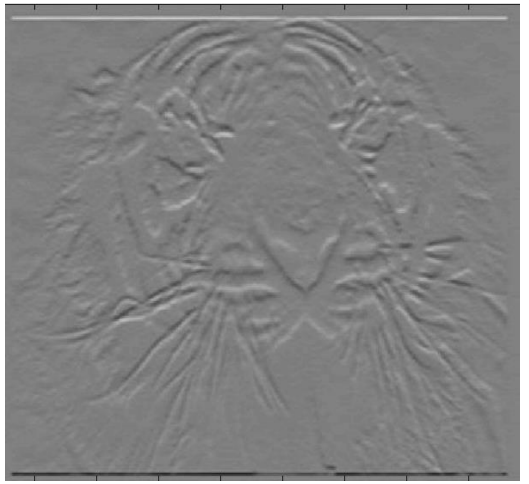
Roberts: $M_x =$

0	1
-1	0

; $M_y =$

1	0
0	-1

```
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```



Laplacian of Gaussian (LoG)

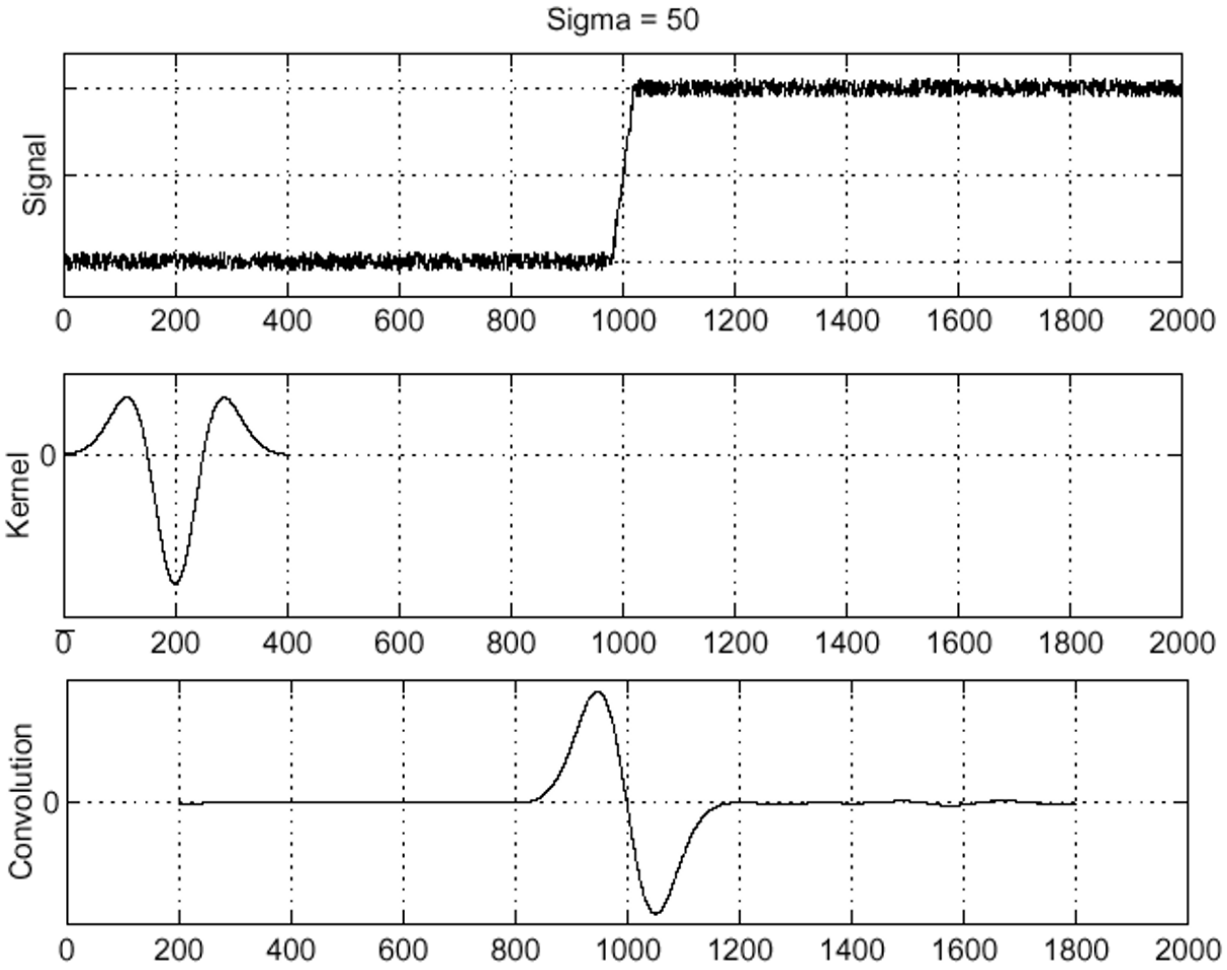
- How does this function $\frac{\partial^2}{\partial x^2}(h \star f)$?

h ... Gaussian

f

$\frac{\partial^2}{\partial x^2} h$

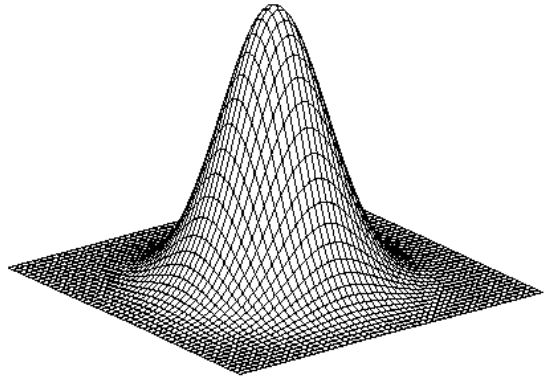
$(\frac{\partial^2}{\partial x^2} h) \star f$



Where's the edge?

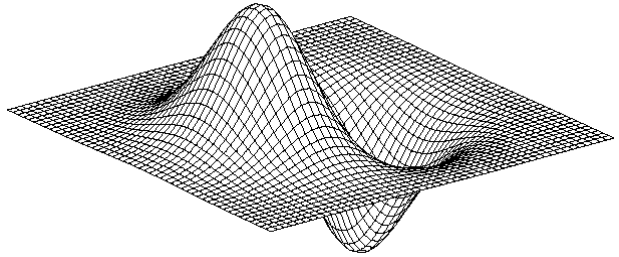
Where the function intersects with x (y=0) (**Zero-crossing**)

Summary: 2D filters for edges



Gaussian

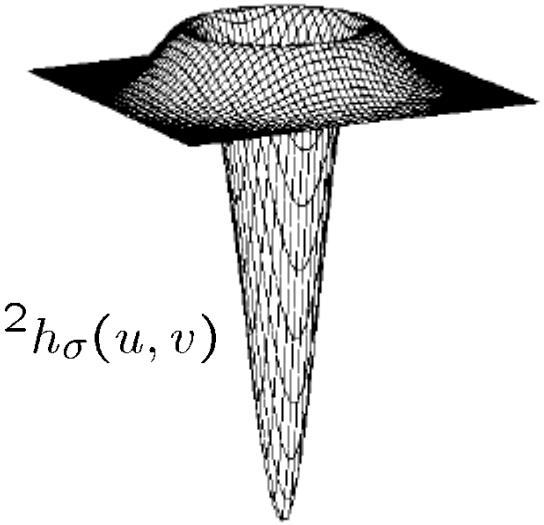
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



Gaussian derivative

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Laplacian of Gaussian

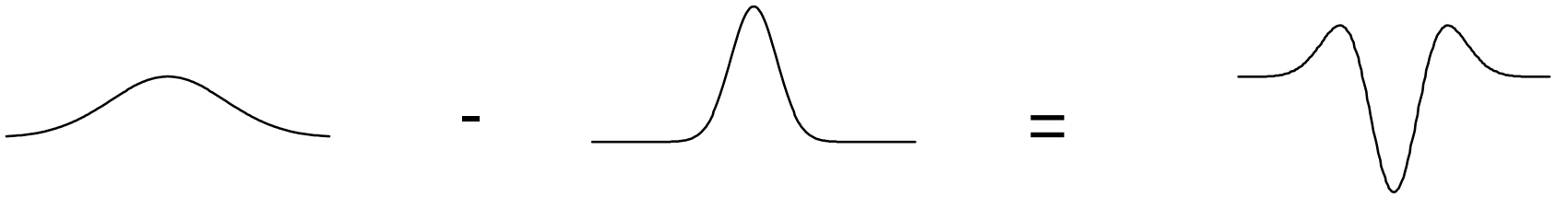


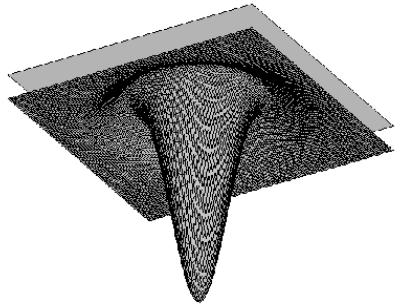
$$\nabla^2 h_\sigma(u,v)$$

- ∇^2 is a Laplace operator (a Laplacian):

$$\nabla^2 h = \frac{\partial^2 h}{\partial u^2} + \frac{\partial^2 h}{\partial v^2}$$

Laplacian ≈ Difference of Gaussians





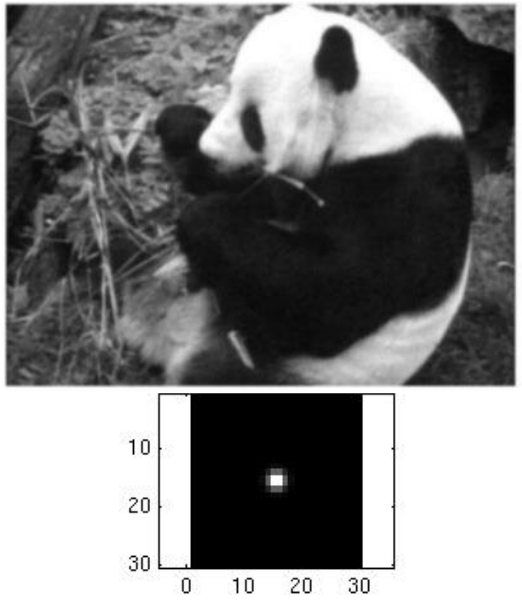
DoG = Difference of Gaussians

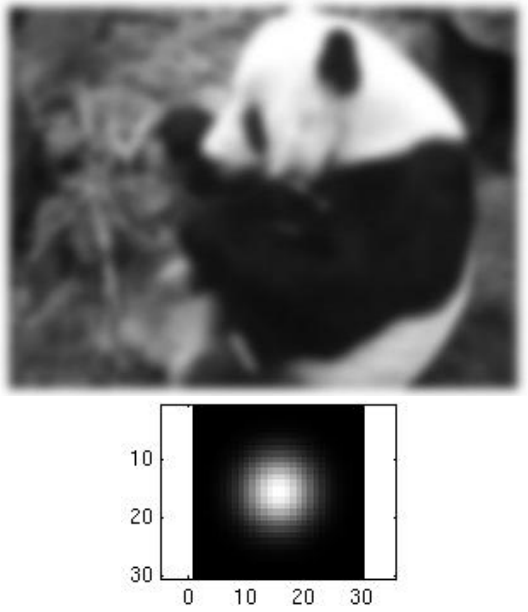
Fast approximation, convolution with a Laplacian kernel.

Edge scale influenced by filters

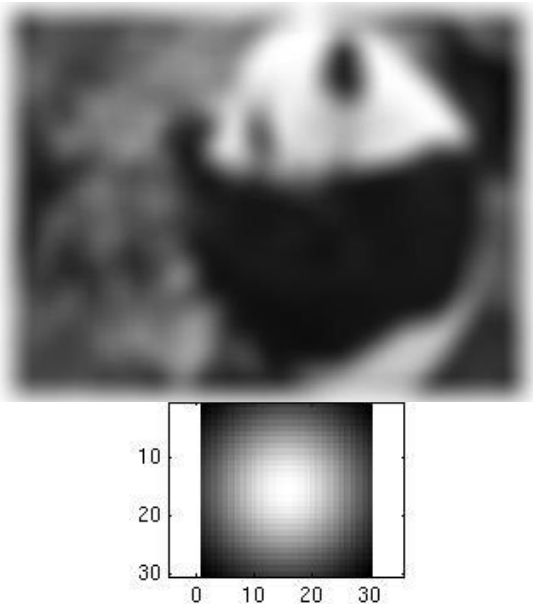
Recall:

Parameter σ is the “scale”/“width” of a Gaussian kernel that determines the extent of smoothing.



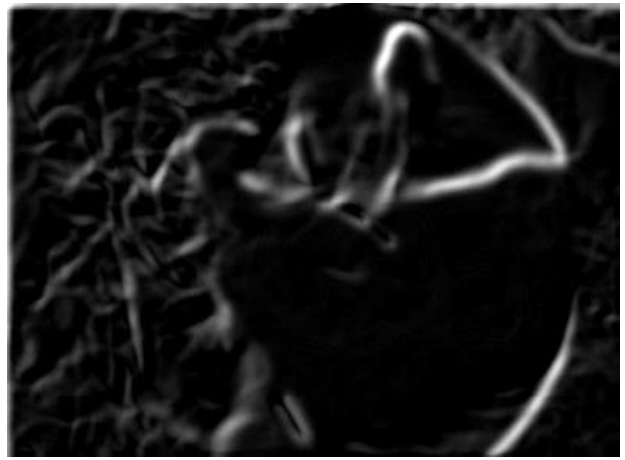


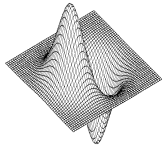
...



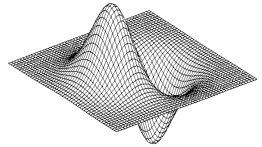
Edge scale influenced by filters

How does σ affect the derivative?





σ = 1 pixel



σ = 3 pixels

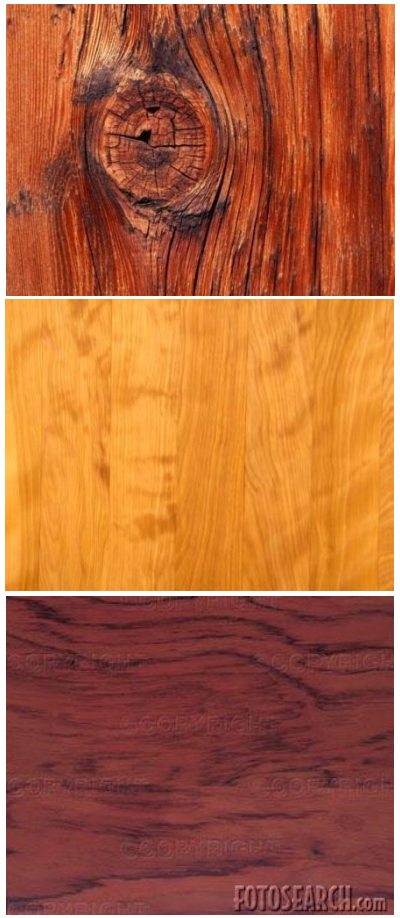
The enhanced/detected structures depend on the Gaussian kernel size.

Large kernels: detect edges on a larger scale.
Small kernels: detect edges on a smaller scale.

What do we mean by “detect edges”?

Depends on what we’re looking for...

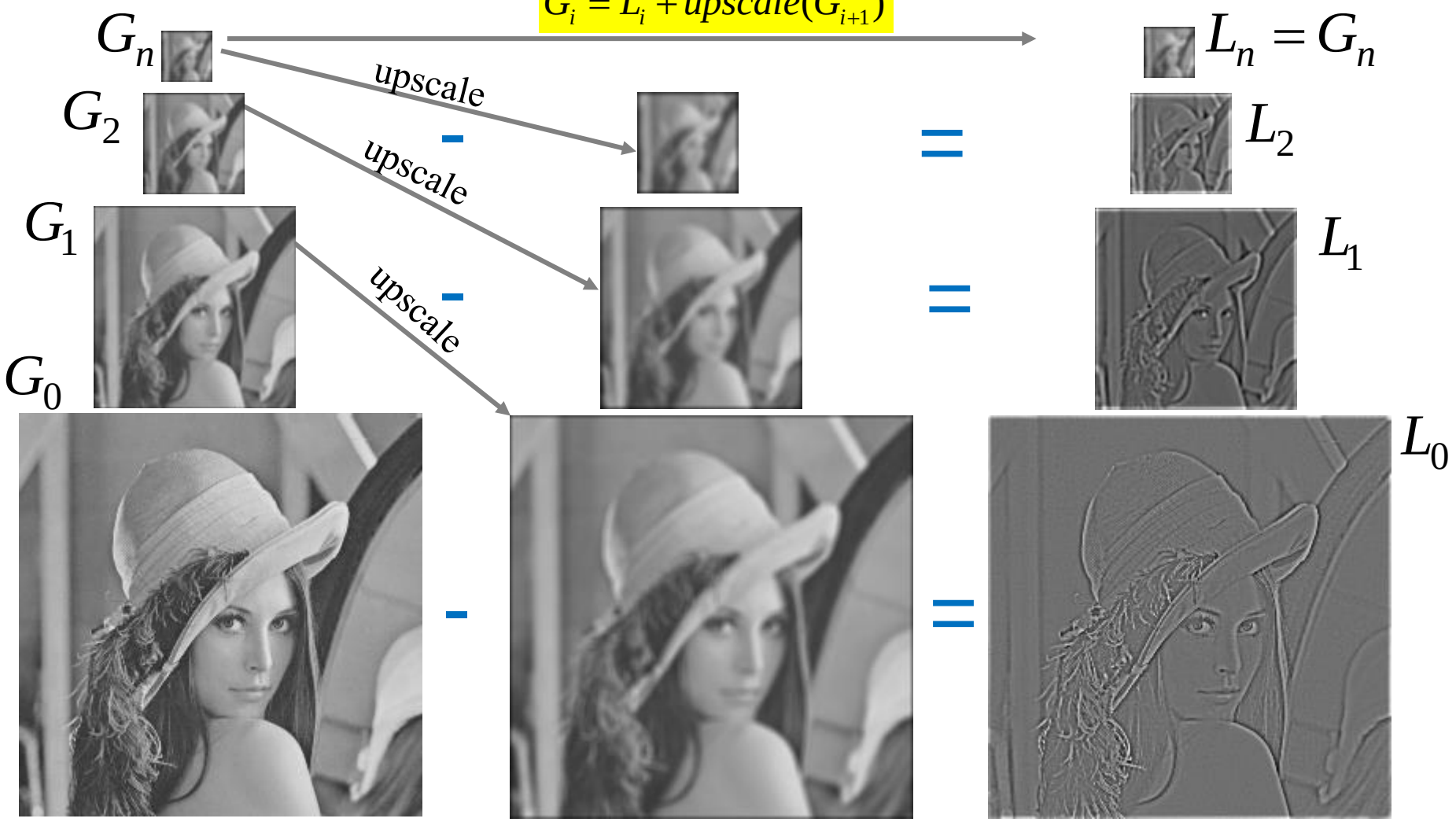
Different structures are found on different scales!



Edge scale and the Laplacian pyramid

$$L_i = G_i - upscale(G_{i+1})$$

$$G_i = L_i + upscale(G_{i+1})$$



Source: Irani & Basri

Edge scale and the Laplacian pyramid

L_2



L_1

L_0


- Instead of increasing the kernel size of derivative filter, the image is reduced and "filtered" with a fixed-size filter!
- Thus higher levels "respond" to thicker edges.

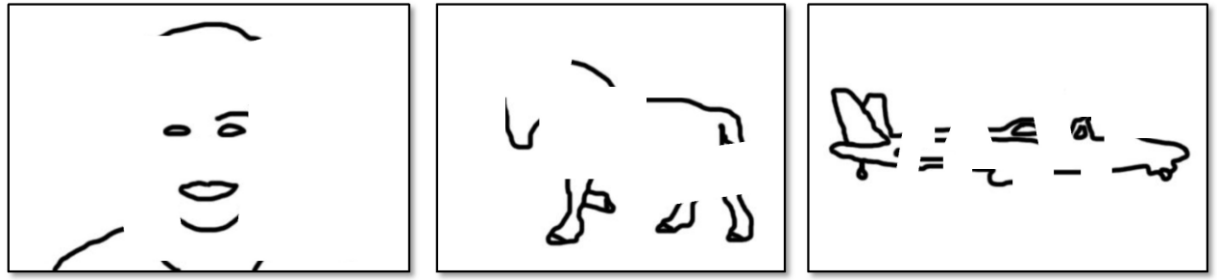
Machine perception

FROM DERIVATIVES TO EDGE DETECTION

The task of edge detection

- Goal: map image from 2D grayscale intensity pixel array into a set of binary curves and lines.
- Why?

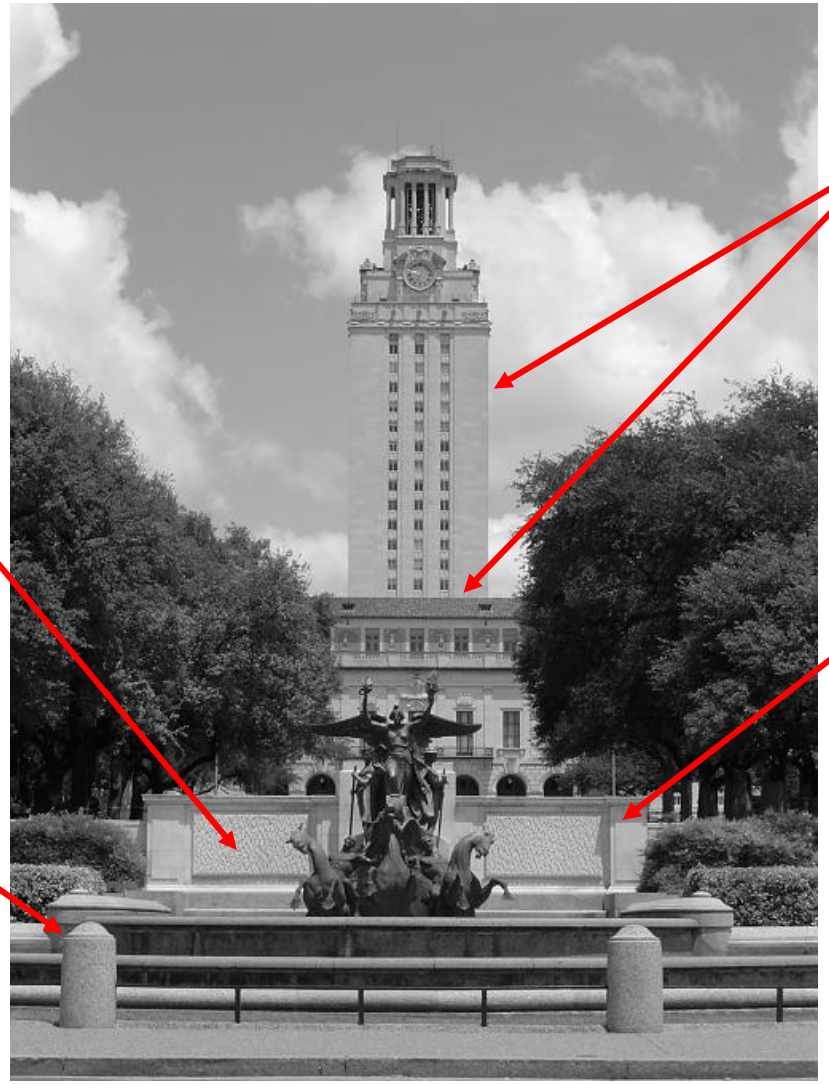
abstraction



Robust, compact representation

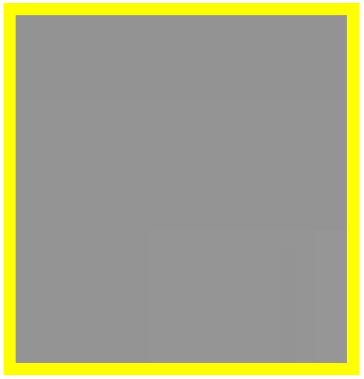
What constitutes an edge?

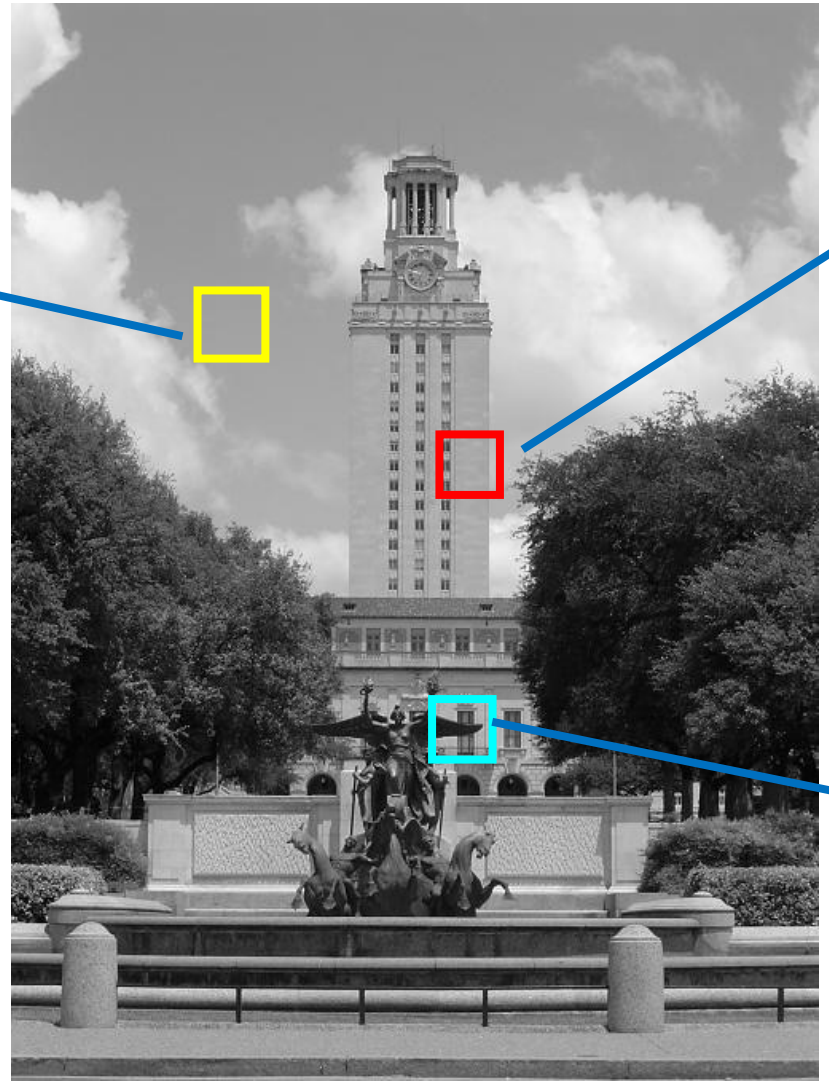
Anything that *appears* as an edge…

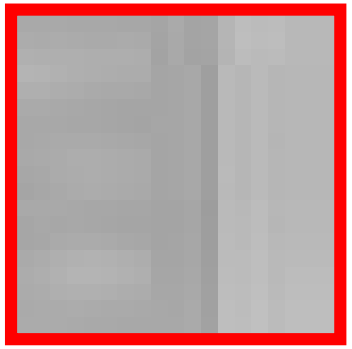


What constitutes an edge?

Anything that *appears* as an edge…

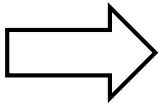






Edge presence is strongly correlated with the local intensity changes.

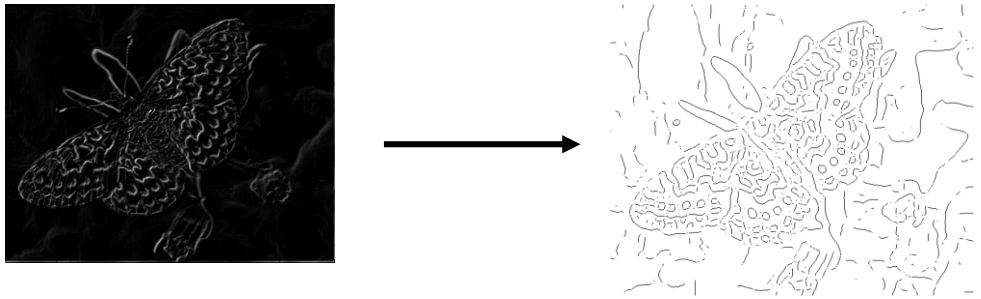
The task of edge detection



- Basic approach:

  find strong gradients + post process

From gradients to edges



Three steps for edge detection

1. Smoothing:

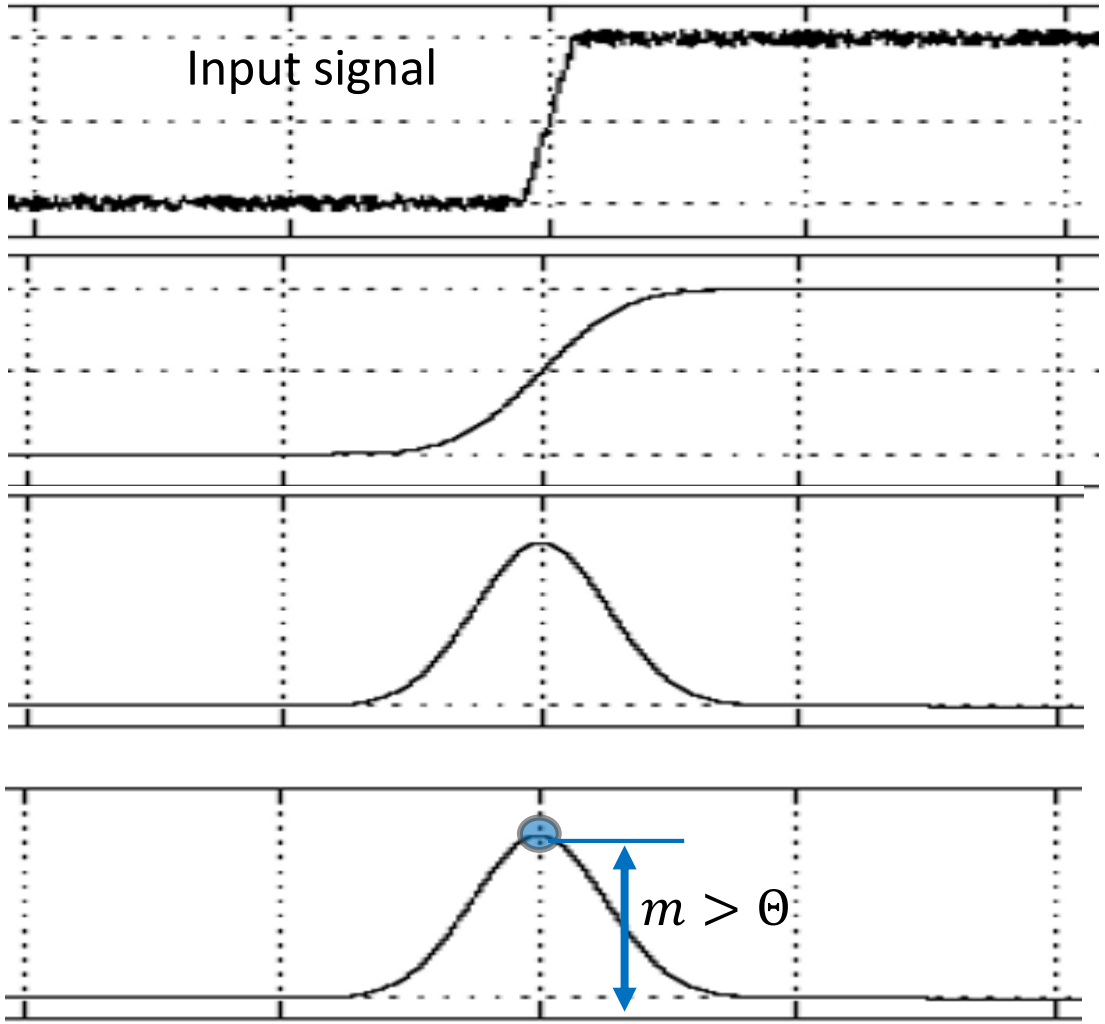
   noise suppression

2. Edge enhancing:

   increase contrast (by differentiating)

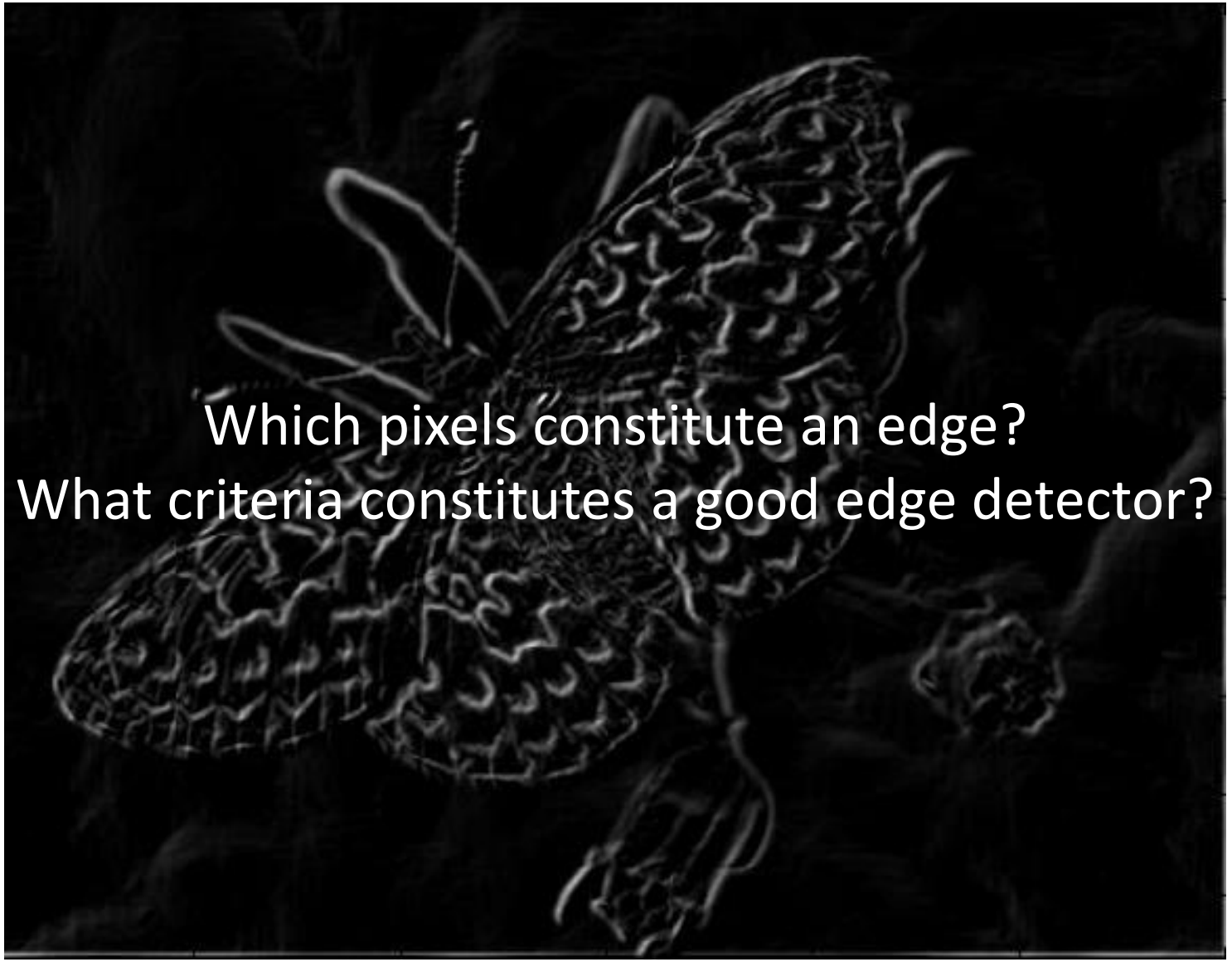
3. Edge localization

   - Identify potential edge pixels
   - Verify the magnitude



Original image

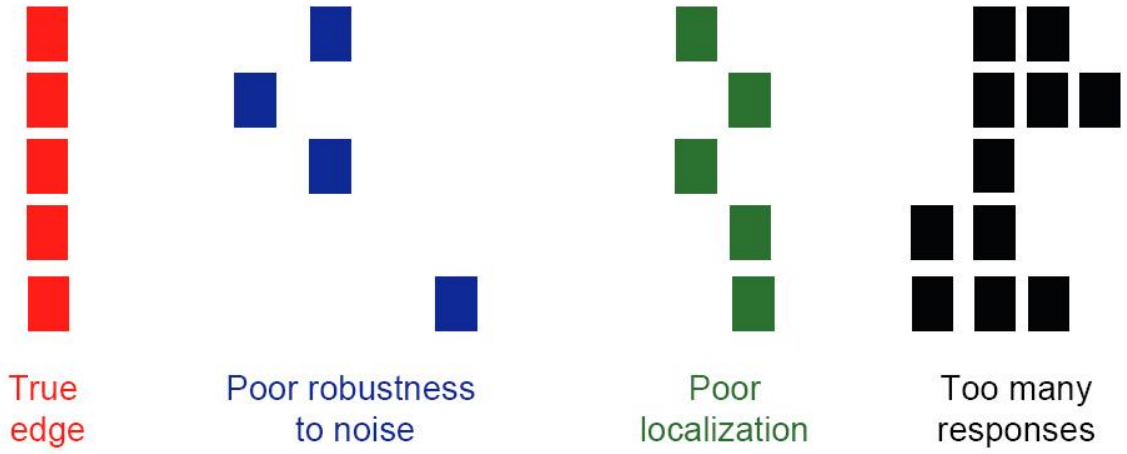
Gradient magnitudes (smooth+enhance)



Slide credit: Kristen Grauman

Designing an edge detector...

- Criteria of “optimal” edge detector:
  1. **Good detection:** optimal detector minimizes probability of false positives (edges caused by noise), and false negatives (missing true edges)
  2. **Good localization:** detected edges should be close to the location of the true edges.
  3. **Specificity:** detector should return only a single point per true edge; minimize number of local maxima around true edge.



Source: Li Fei-Fei

The Canny edge detector

- Most popular edge detector in computer vision.
- Theoretical model:

  Step edges with added Gaussian noise.
- Canny showed that first derivative of a Gaussian well approximates an operator that optimizes a tradeoff between signal-to-noise ratio and localization.

MATLAB:

```
>> edge(image, 'canny');
>> help edge
```

J. Canny, *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Canny edge detector

1. Filter image by a derivative of a Gaussian
2. Calculate the gradient magnitude and orientation
3. Non-maxima suppression:
   - Convert multiple pixels-thick edges to a single-pixel thickness: Thinning
- ….

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Source: Li Fei-Fei

Canny edge detector: Thinning

Original image (Lena)

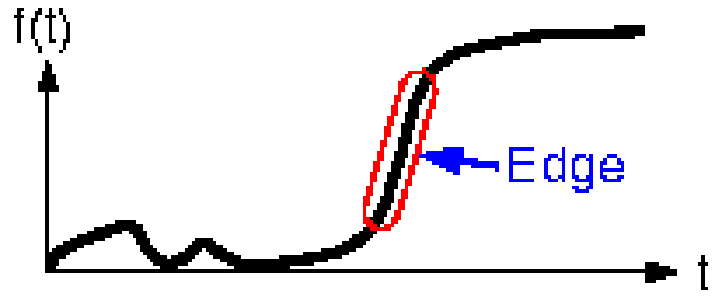
Canny edge detector: Thinning – 1. Derivative

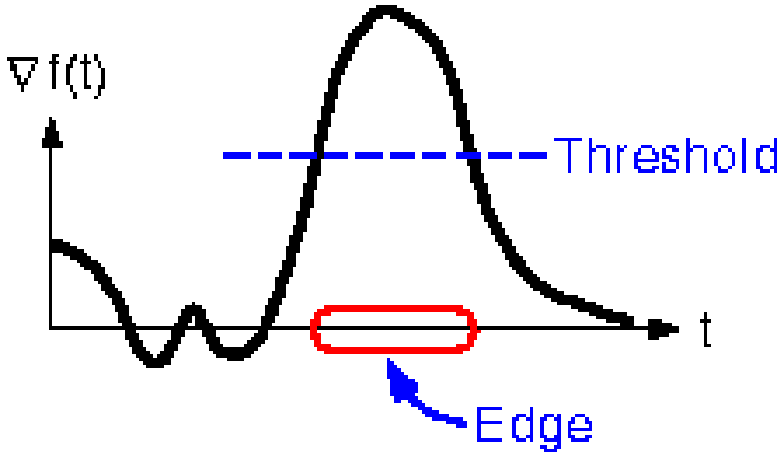
Gradient magnitude

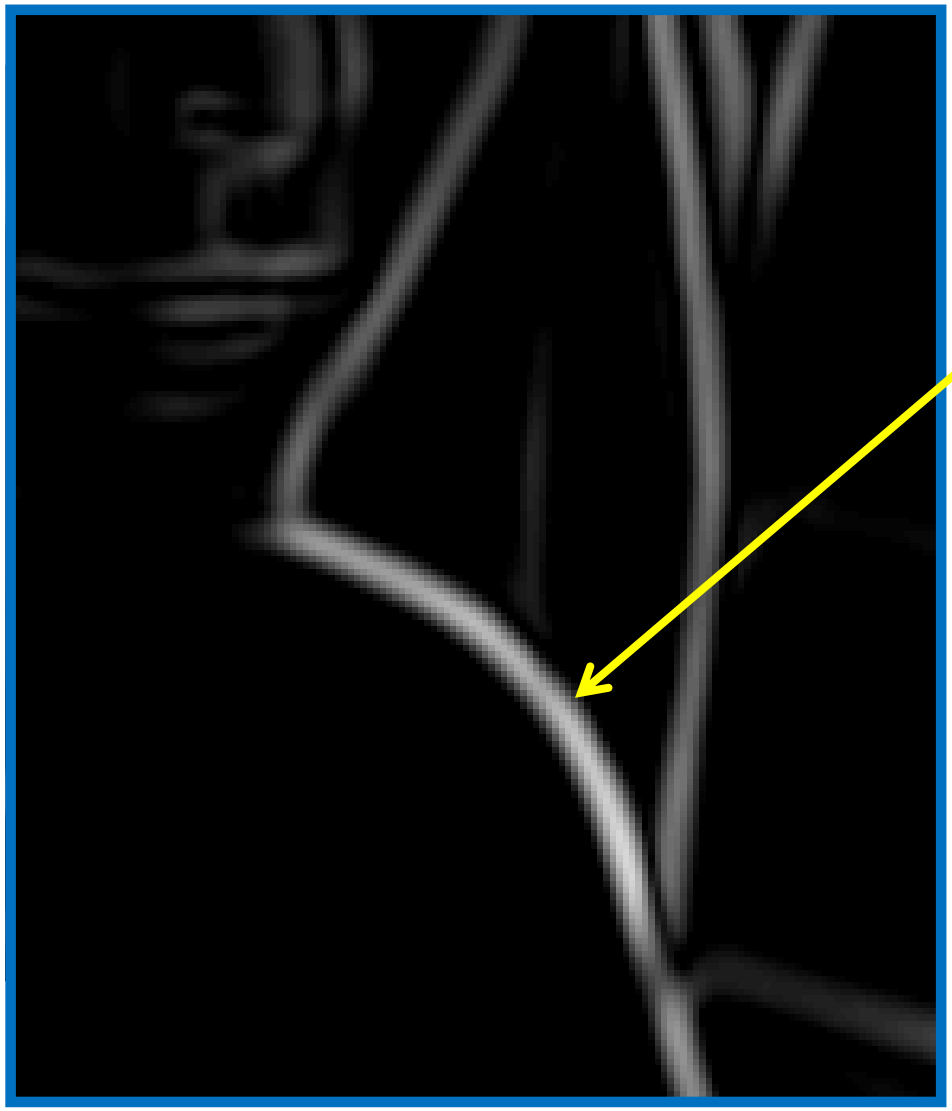
Canny edge detector: Thinning - 2. Threshold

„Thresholding“: Set magnitudes lower than a prescribed threshold to 0.

Canny edge detector: Thinning

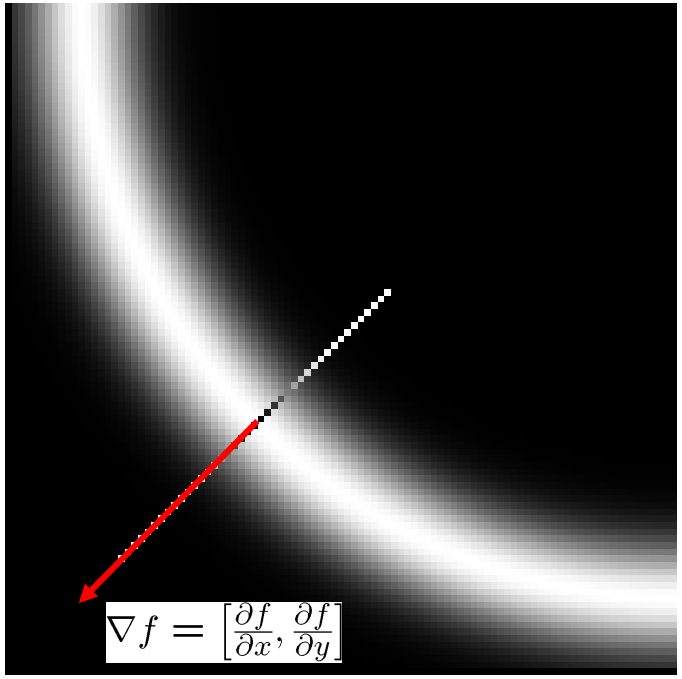


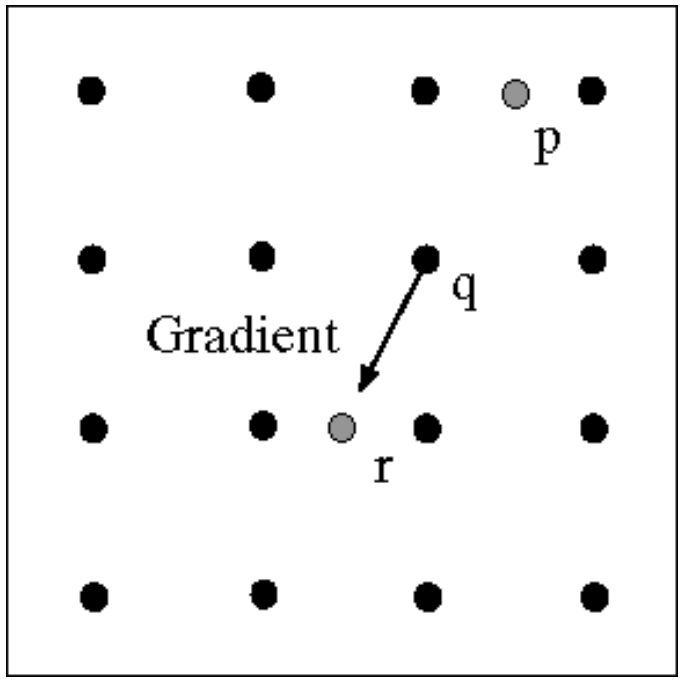




How to convert these thick lines into thinner curves?

Thinning by non-maxima suppression





- For each pixel check if it is a local maximum along its gradient direction.
- Only local maxima should remain.
  - Advanced: Actually, for **q**, we should check interpolated pixel values at **p** and **r**.

Canny edge detector: Thinning – 3. Nonmax Suppression

Thinning

(non-maximum suppression)

Canny edge detector: Thinning

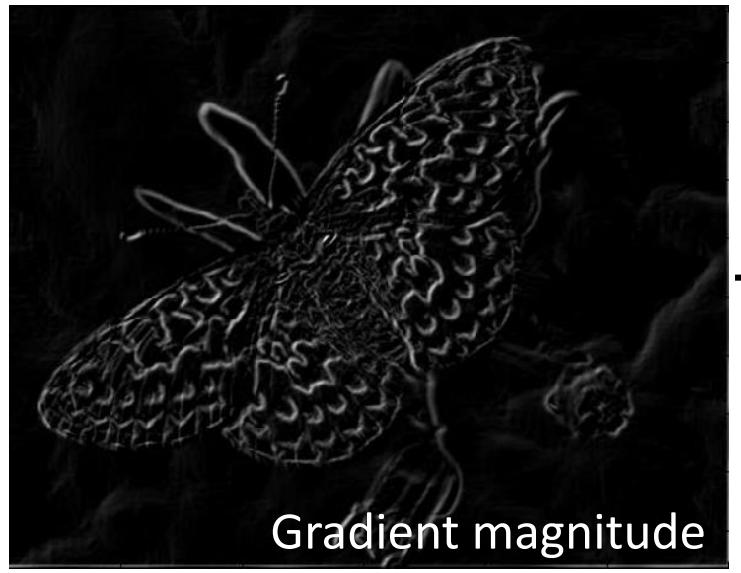
Problem: pixels along this edge did not „survive“ the thresholding.

Thinning

(non-maximum suppression)

How to select a threshold?

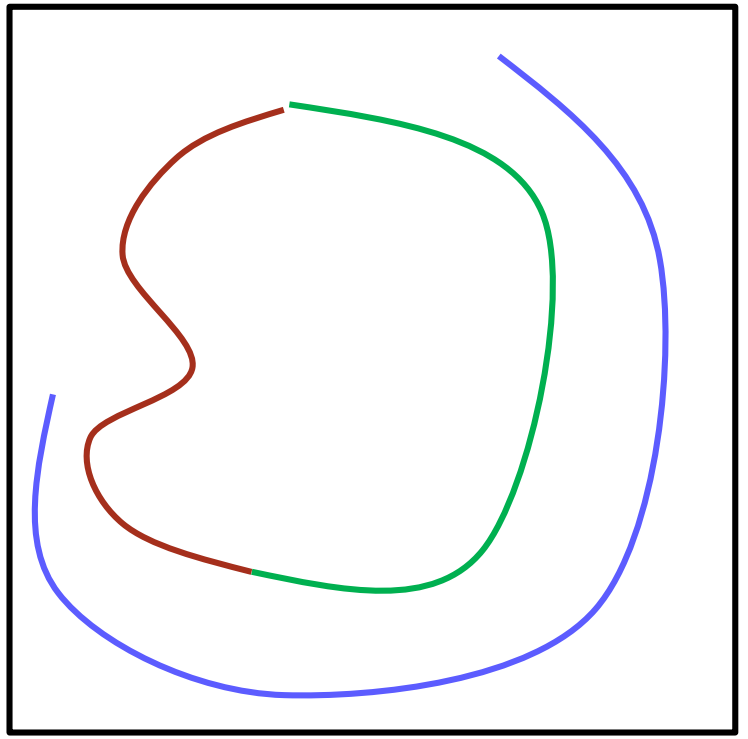
Thinning

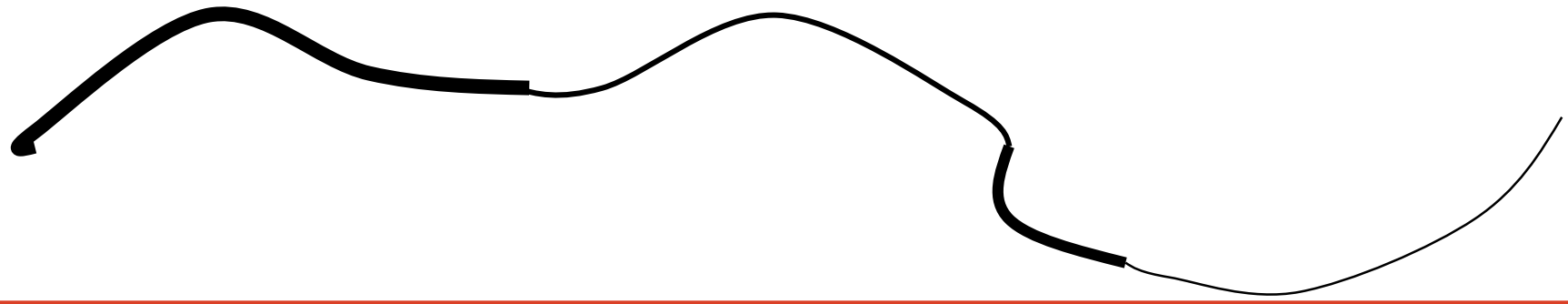


Thinning

Canny edge detector: Hysteresis thresholding

- Trace each contour separately (e.g., using 4-connectedness).
- Apply two thresholds k_{high} and k_{low}
  - Start tracing a line only at pixels that pass the k_{high} threshold.
  - Continue tracing if the pixels pass k_{low} threshold.
- Typical threshold ratio: $k_{high} / k_{low} = 2$





Hysteresis thresholding

Original

High threshold
(strong edges)

Low threshold
(weak edges)

Hysteresis thresholding

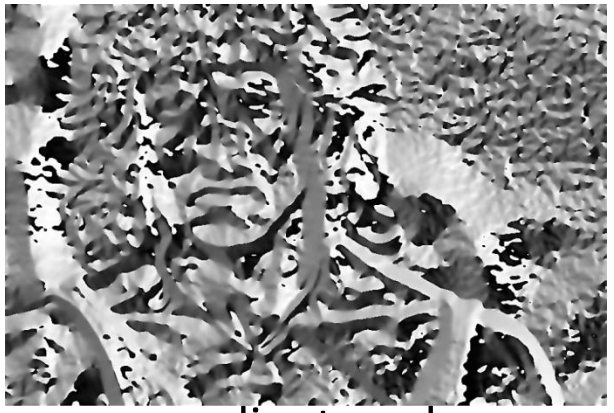
The Canny edge detector in a nutshell

1. Convolve the image by a derivative of a Gaussian.
2. Calculate the gradient magnitude and orientation
3. Non-maxima suppression
   - Thin edges to one-pixel width.
4. Trace the edges by hysteresis thresholding
   - Apply a high threshold on the magnitude to initialize a contours and continue tracing the contour until the magnitude falls below a low threshold.

Canny edge detector in “*action*”

Input image

gradient magnitude



gradient angle

thinned

Thresholded by hysteresis

Canny edge detector in "*action*"

Input image

gradient magnitude

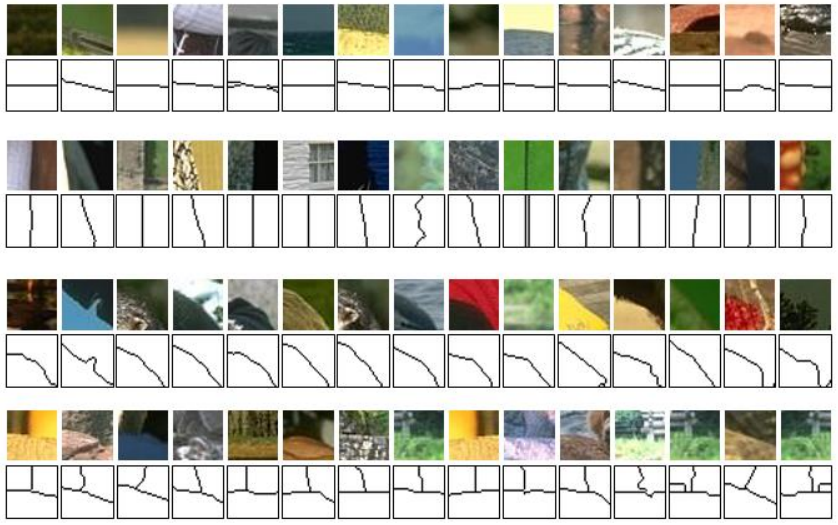
gradient angle

thinned

Thresholded by hysteresis

Beyond Canny edge detector

- Recently lots of new approaches for edge detection by machine learning.
- Essentially, look at patches and learn what an edge is by inferring the structure from intensities.



Sketch Tokens, CVPR 2013. Joseph Lim, C. Zitnick, and P. Dollár

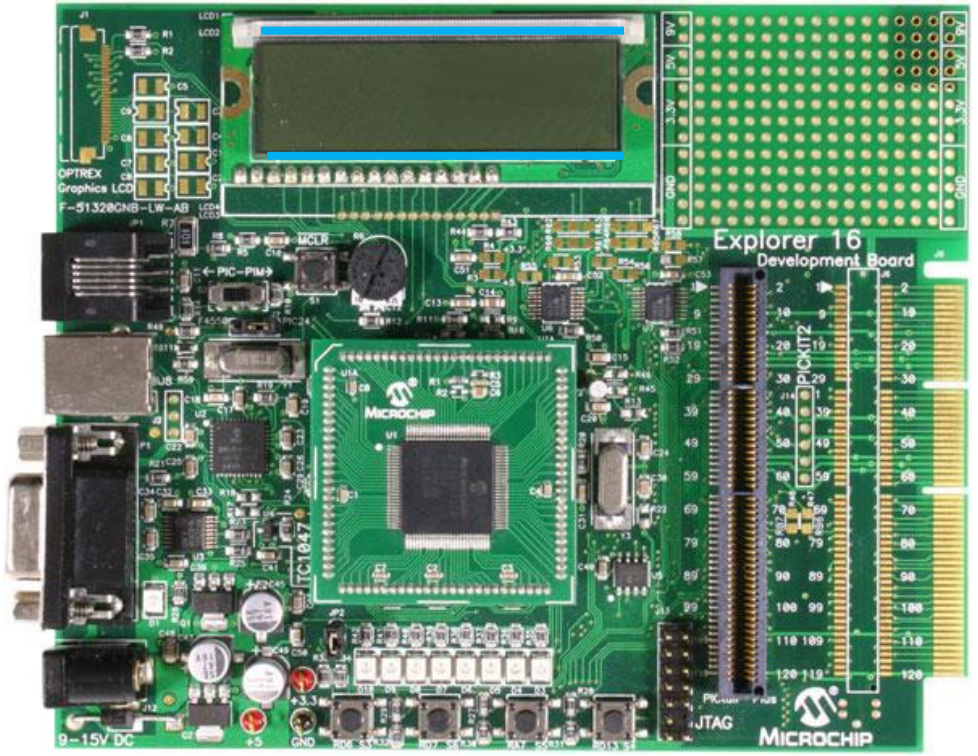
Machine perception

EDGE DETECTION USING PARAMETRIC MODELS

Example: line fitting

- Why should we fit lines?

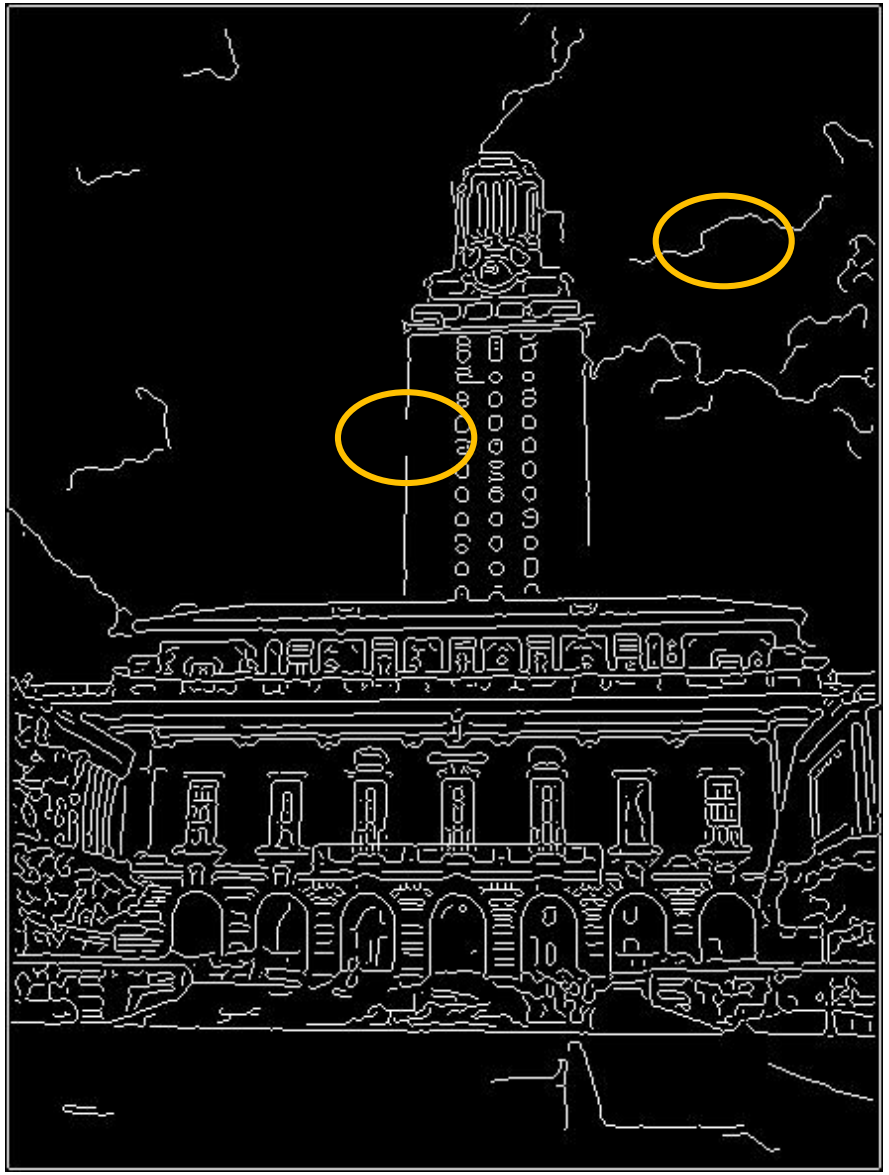
  Many scenes are composed of straight lines



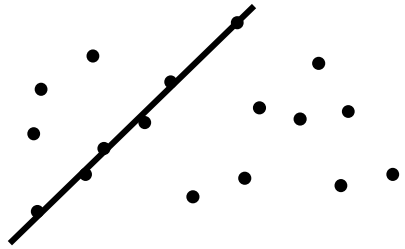
- But, haven't we done that already by edge detection?

Slide credit: Kristen Grauman

Problems with line fitting



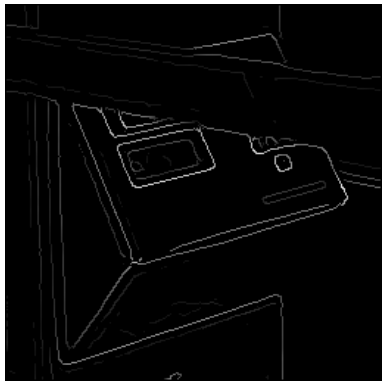
- Noisy edges, multiple models:
  - Which points correspond with which line, if at all?
- Some parts of lines are not detected:
  - How to find a line that connects the missing points?
- Noisy orientation:
  - How do we determine the unknown parameters of true lines?

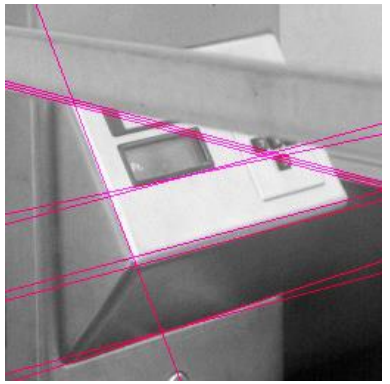


Line fitting by voting for parameters

- Given a set of points, find the lines.
- How many lines?
- Which points correspond to which lines?

- *Hough Transform* is a voting technique that answers these questions.
- Main idea:
  1. For each edge point compute all possible lines that pass through that point.
     - Each point casts a vote for parameter values.
  2. Select the lines (parameter combinations) that receive enough votes.





Hough space: straight lines

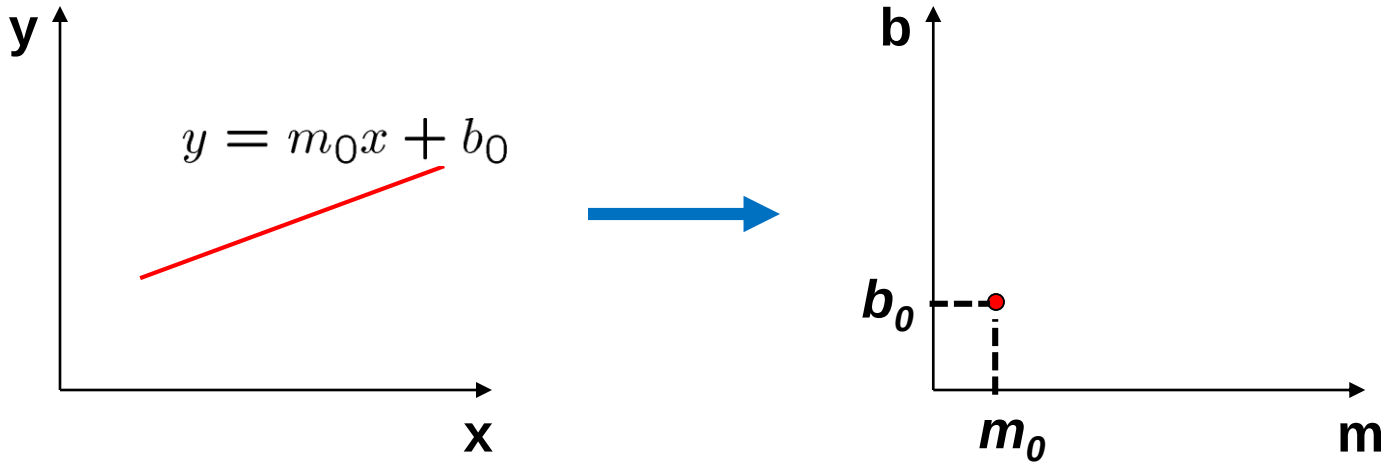
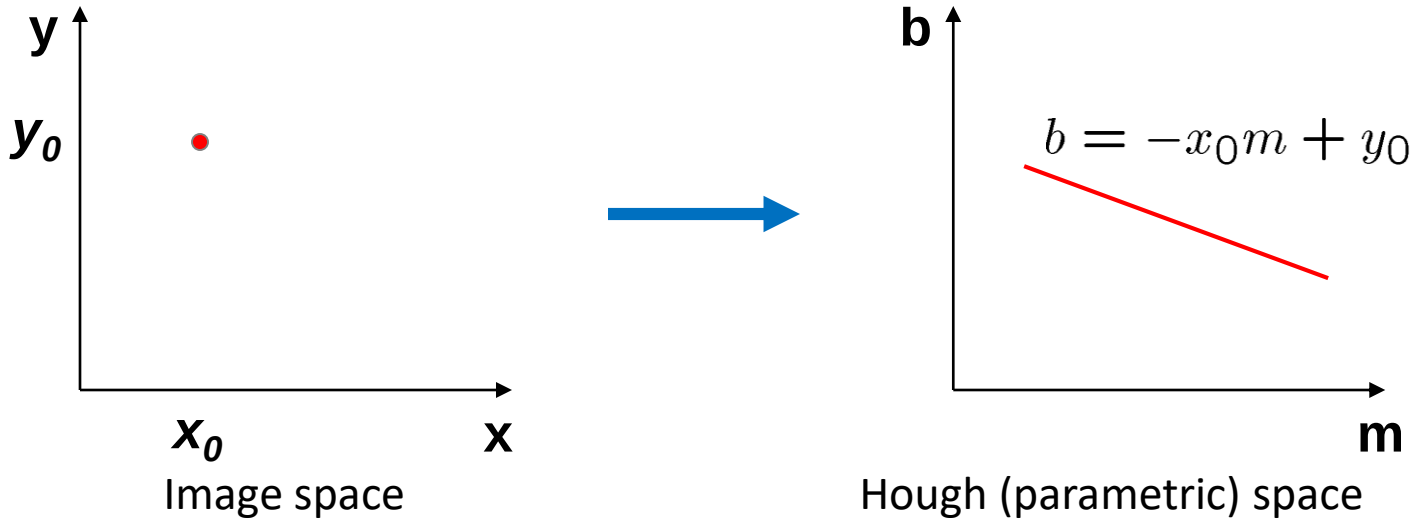


Image space

Hough (parametric) space

- Connection between spatial (x,y) and Hough space (m,b):
  - A line in image corresponds to a point in the Hough space.

Hough space: straight lines



- Connection between spatial (x,y) and Hough space (m,b):
  - A line in image corresponds to a point in the Hough space.
  - Mapping from image to Hough space:
    - For a point (x,y), find all (m,b) for which this holds : $y = mx + b$

Hough space: straight lines

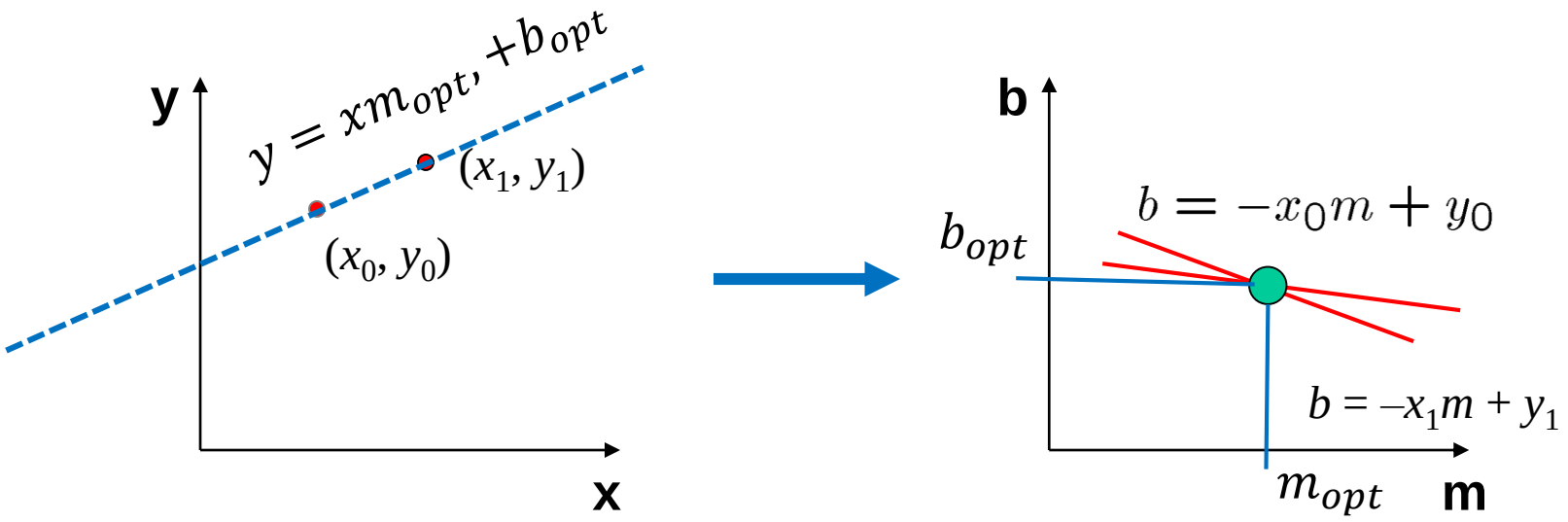


Image space

Hough (parametric) space

- Connection between spatial (x,y) and Hough space (m,b):
  - A line in image corresponds to a point in the Hough space.
  - Mapping from image to Hough space:
    - For a point (x,y), find all (m,b) for which this holds : $y = mx + b$

Hough space: straight lines

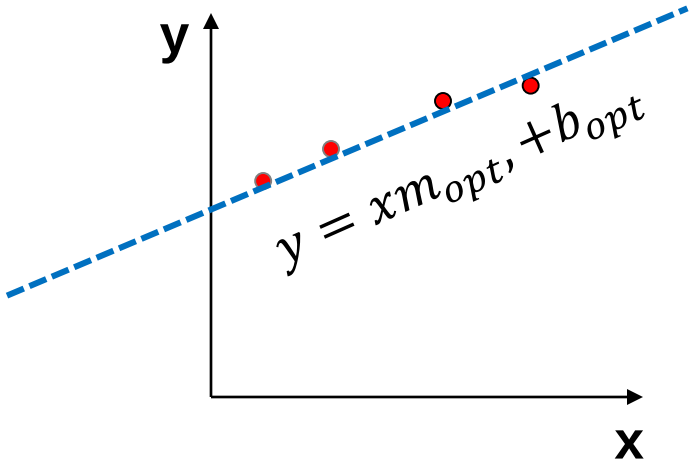
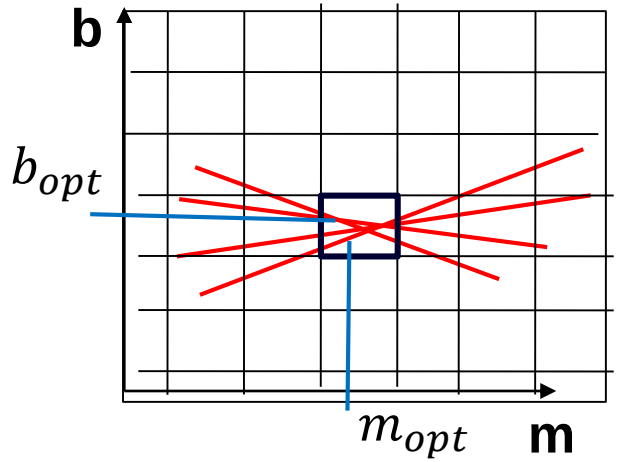


Image space



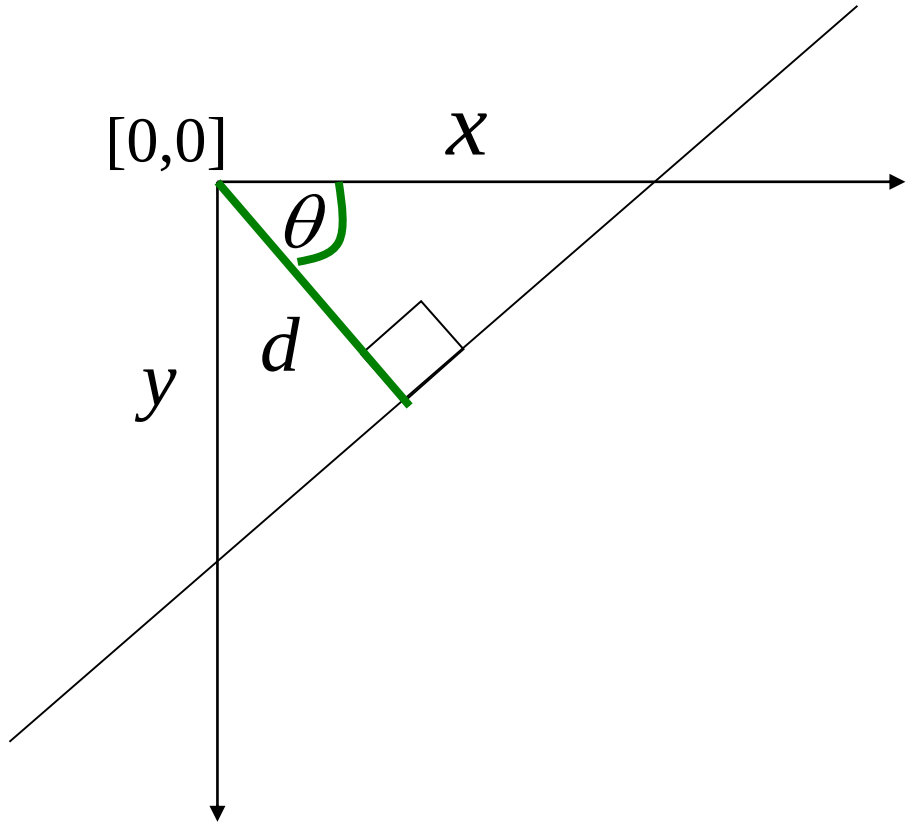
Hough (parametric) space

Discretize the parameter space…

- Connection between spatial (x,y) and Hough space (m,b):
  - A line in image corresponds to a point in the Hough space.
  - Mapping from image to Hough space:
    - For a point (x,y), find all (m,b) for which this holds : $y = mx + b$

Encode the line in polar coordinates

- Issue with Cartesian (m,b): infinite values for vertical lines!



d : perpendicular distance from the origin

θ: angle of perpendicular line with x axis

$$x\cos\theta - y\sin\theta = d$$

- Point in image $\Rightarrow$ sinusoid in Hough space

Algorithm: Straight lines

Using polar representation:

$$x\cos\theta - y\sin\theta = d$$

Basic Hough transform:

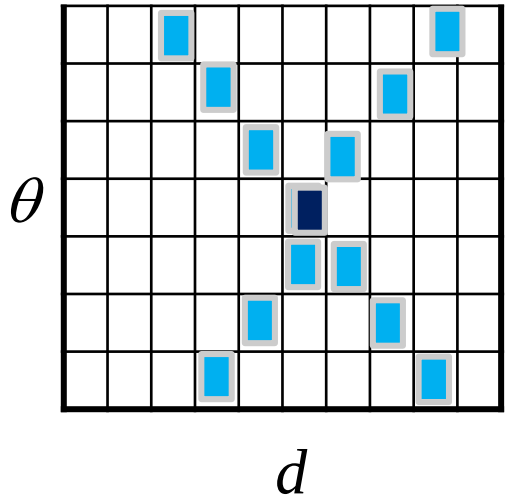
1. Initialize $H[d,\theta] = 0$.
2. For each edge point (x,y) in image

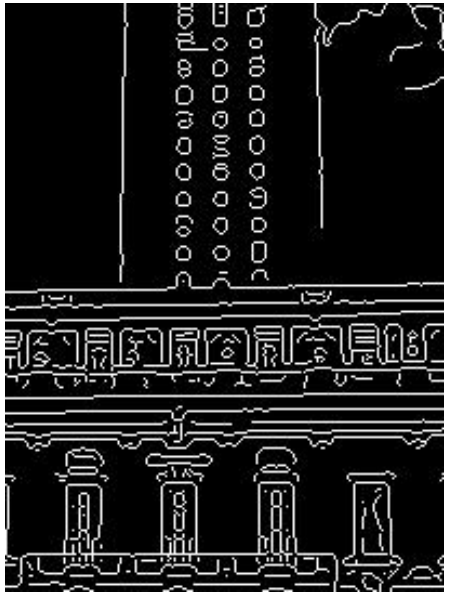
   For $\theta = 0$ to 180 // over quantized values!!

   $$d = x\cos\theta - y\sin\theta$$

   H[d, θ] += 1
3. Find local maxima (d,θ) in accumulator array $H[d,\theta]$.
4. Detected line is defined by: $d = x\cos\theta - y\sin\theta$

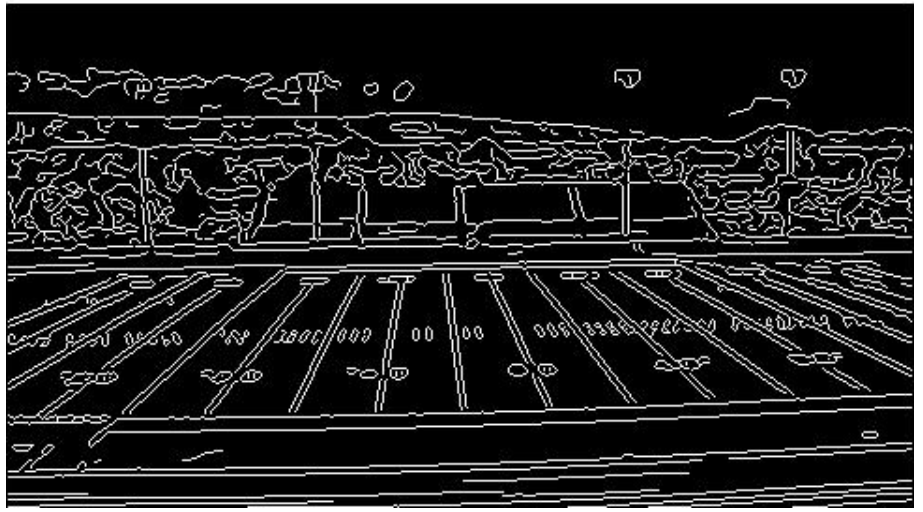
H: accumulator array (votes)

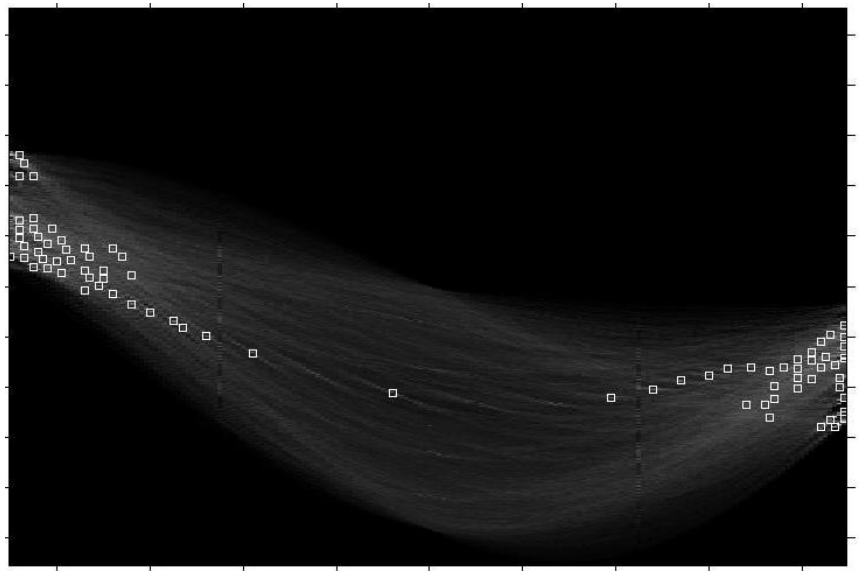


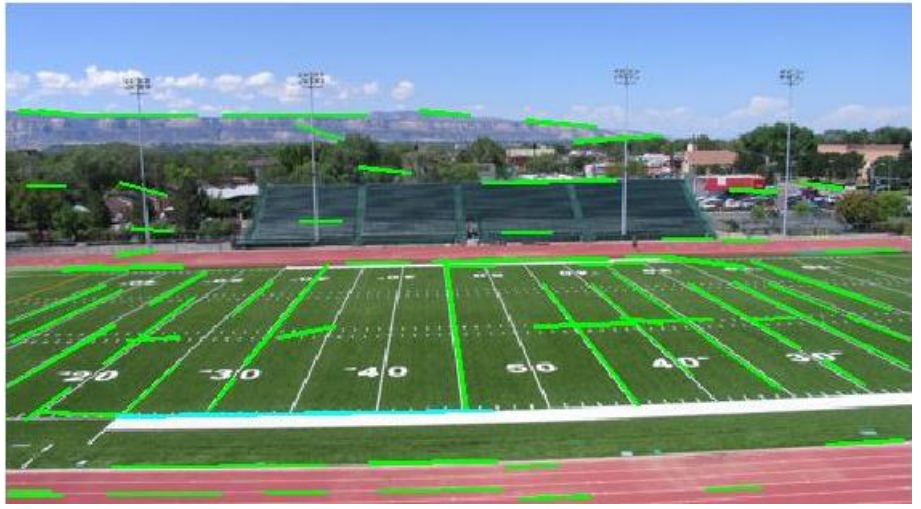


[Hough line demo]

Hough transform in action

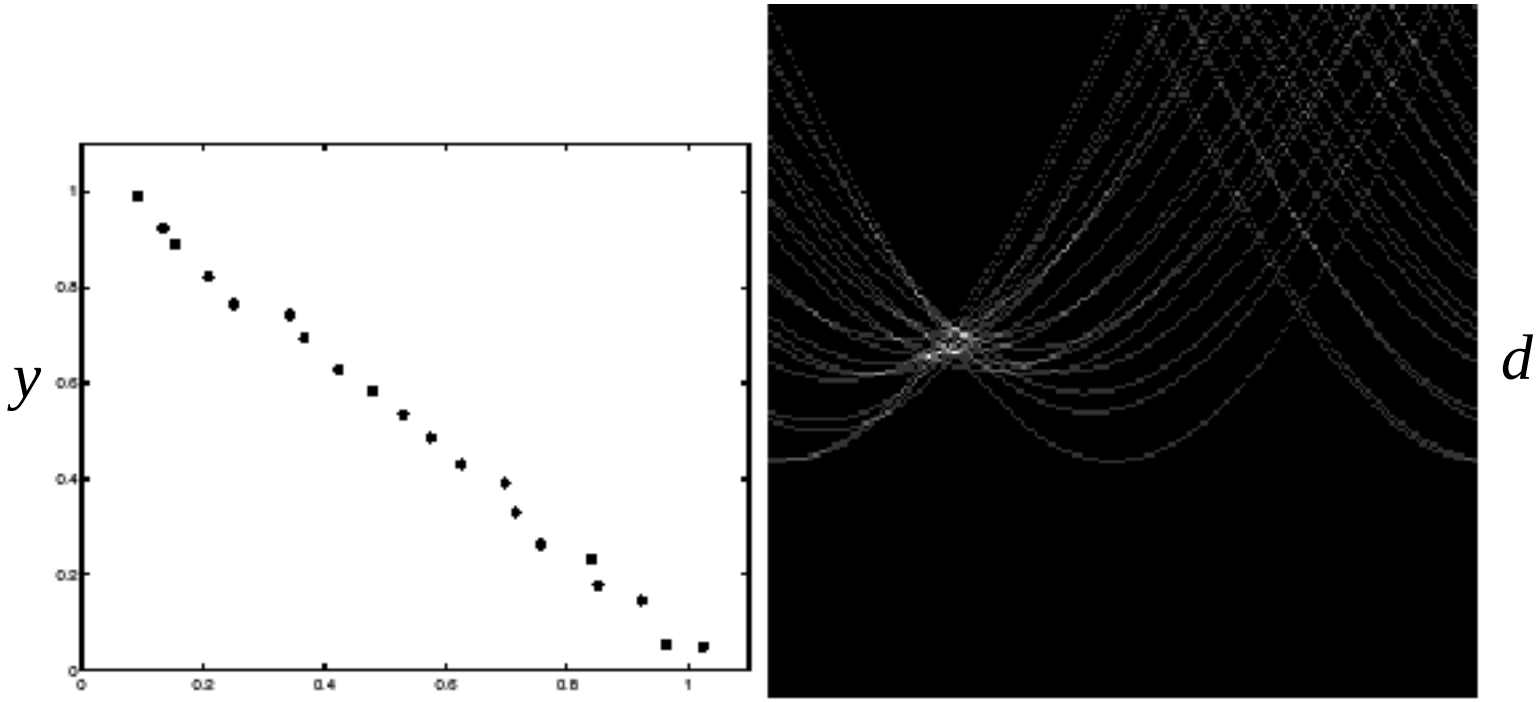






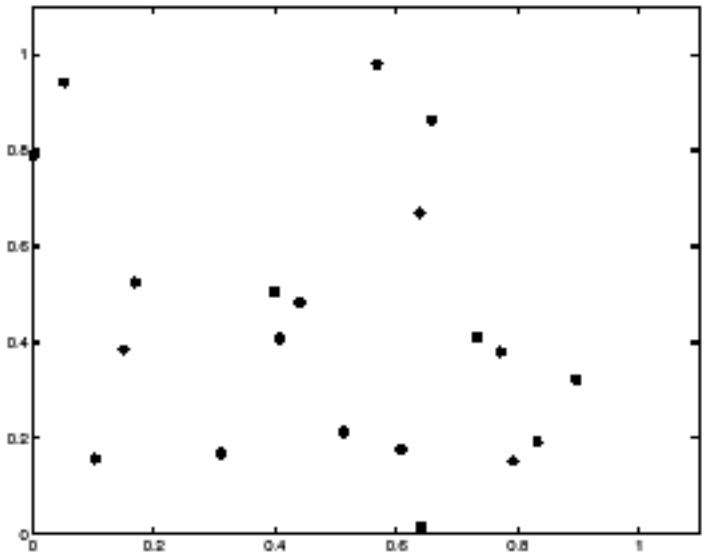
Only the longest segments shown here.

Hough transform: Noise – binning

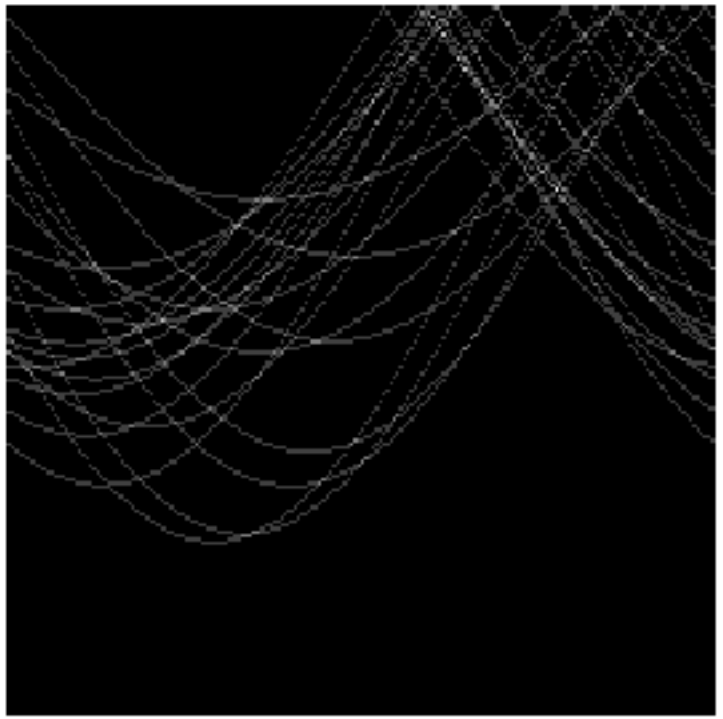


Are there any significant problems with the noise?

Hough transform: Noise – amplitude of votes



Coordinates of
edge points in image



Votes

Random points still form some local maxima in the accumulator array!

Hough transform: Extensions

Extension 1: *Use the gradient info!*

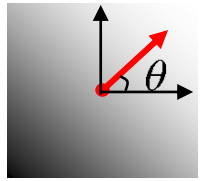
1. *same as standard HT*
2. For each edge point $[x,y]$

   θ = gradient at (x,y)

   $$d = x\cos\theta - y\sin\theta$$

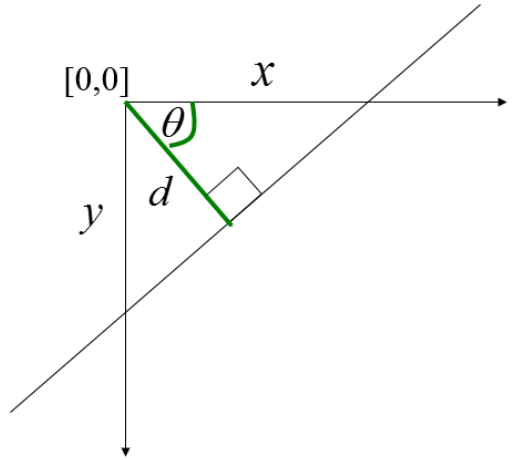
   $$\mathrm{H}[d,\theta] \mathrel{+}= 1$$
3. *same as standard HT*
4. *same as standard HT*

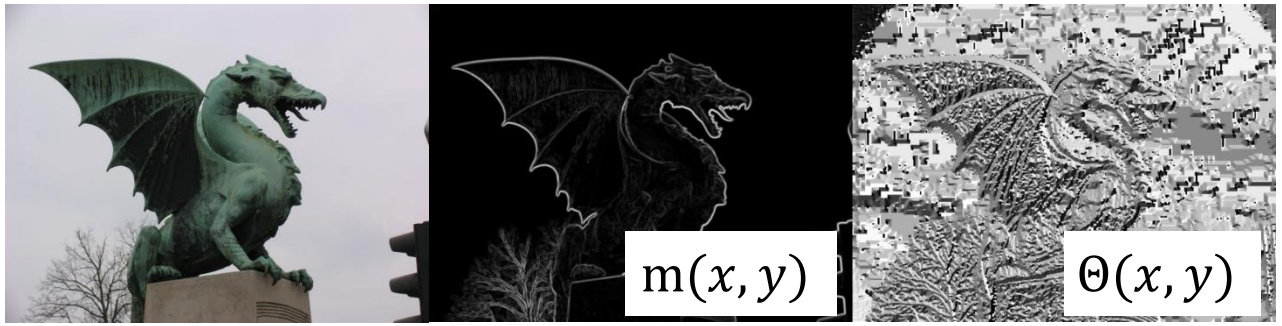
Reduces the number of degrees of freedom (dof)!



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$





Hough transform: Extensions

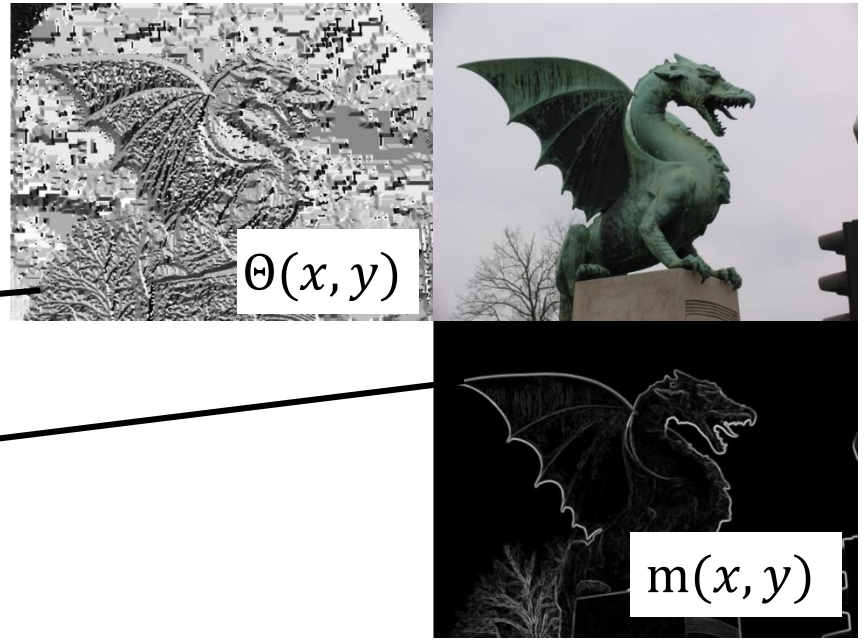
Extension 1: *Use the gradient info!*

1. *same as standard HT*
2. For each edge point $[x,y]$

   θ = gradient at (x,y)

   $$d = x\cos\theta - y\sin\theta$$

   $\mathrm{H}[d,\theta]$ += 1
3. *same as standard HT*
4. *same as standard HT*



Extension 2:

- Assign higher weight in votes to points with large edge magnitude.
  Instead $\mathrm{H}[d,\theta]$ += 1, use $\mathrm{H}[d,\theta]$ += $\mathrm{m}(x,y)$.

- These extensions can be applied in general:

  line, circles, squares, general shapes...

Hough transform for circles

- Circle parameters: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Example of center detection at known radius r

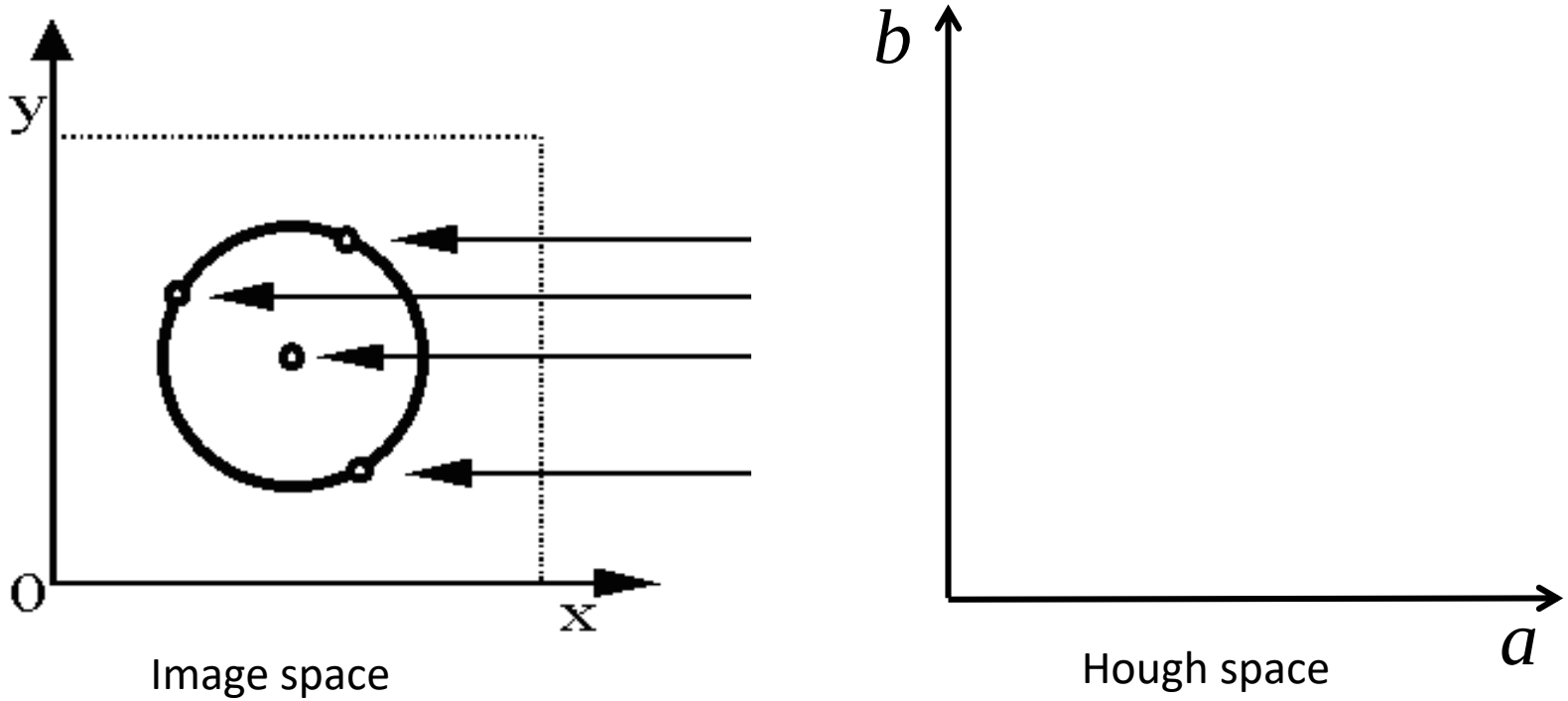


Image space

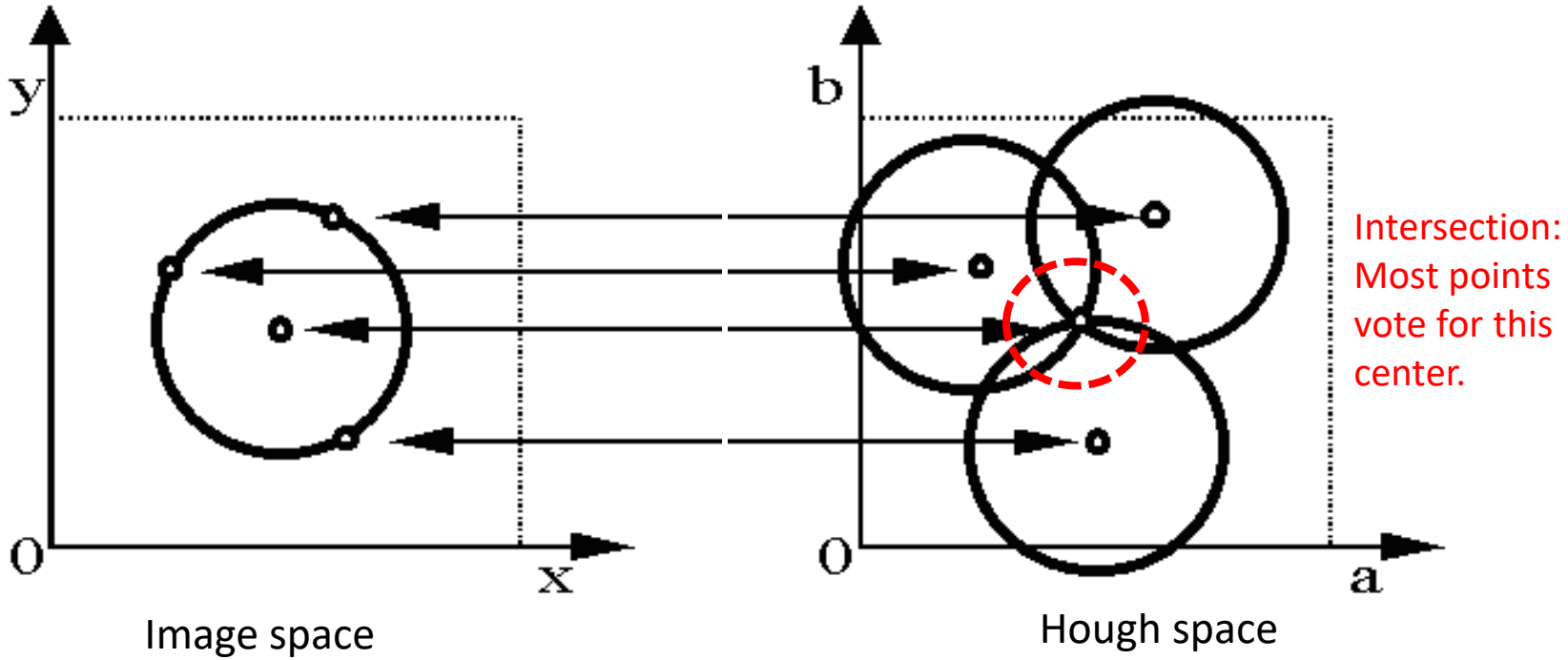
Hough space

Hough transform for circles

- Circle parameters: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Example of center detection at known radius r

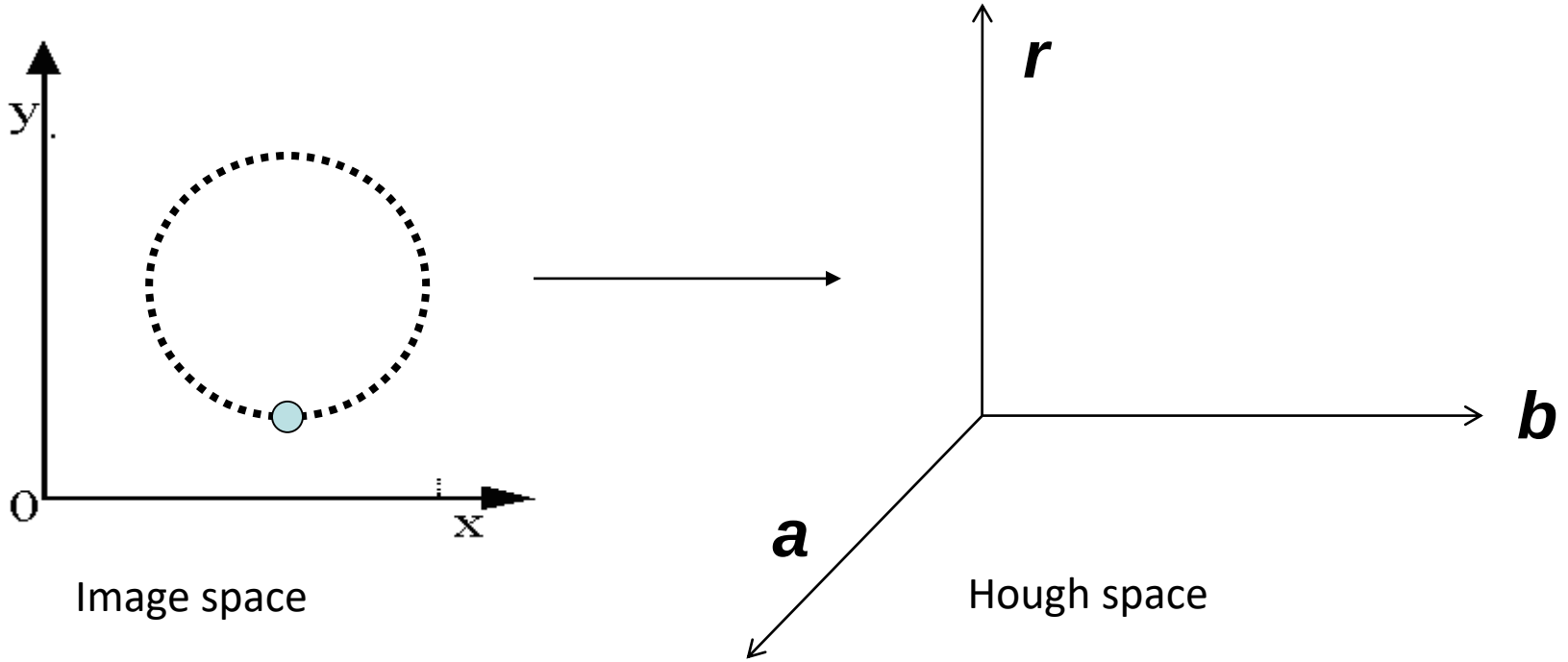


Hough transform for circles

- Circle parameters: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Unknown radius r – *How many dimensions in Hough Space?*



Hough transform for circles

- Circle parameters: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Unknown radius r

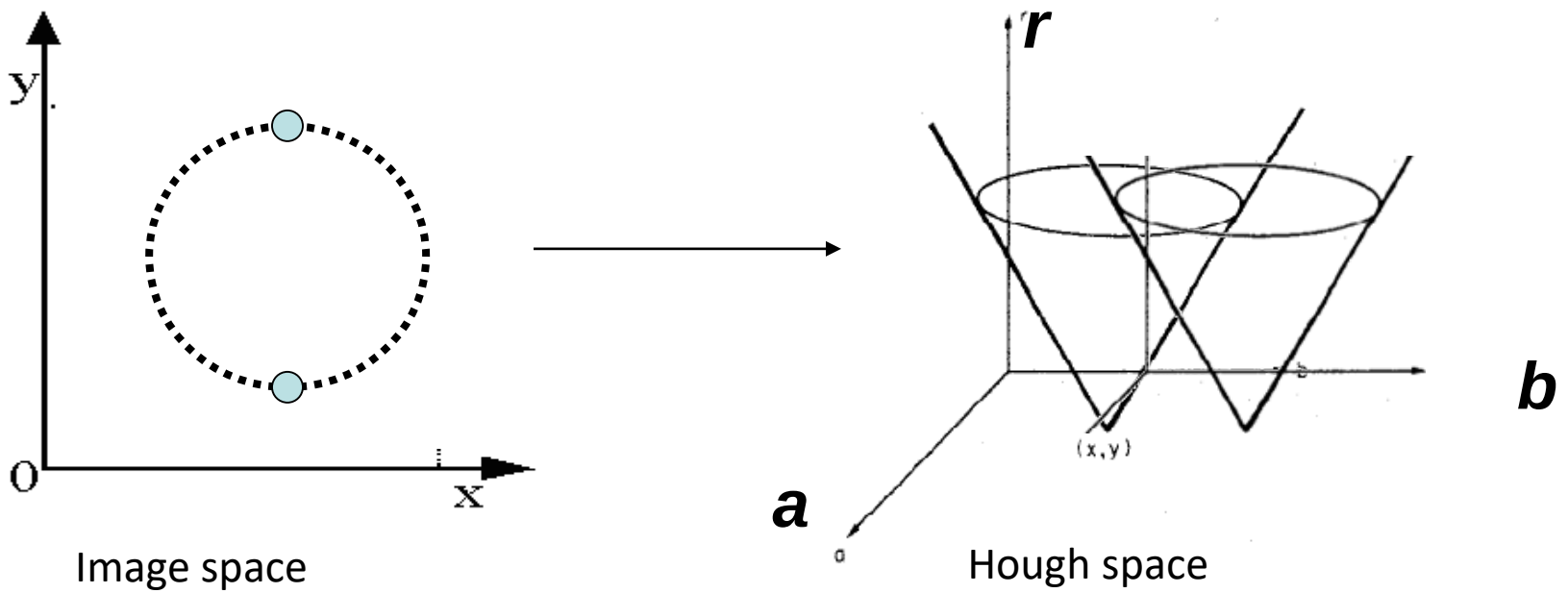


Image space

Hough space

Hough transform for circles

- Circle parameters: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Unknown radius r

But assume we know the gradient direction!

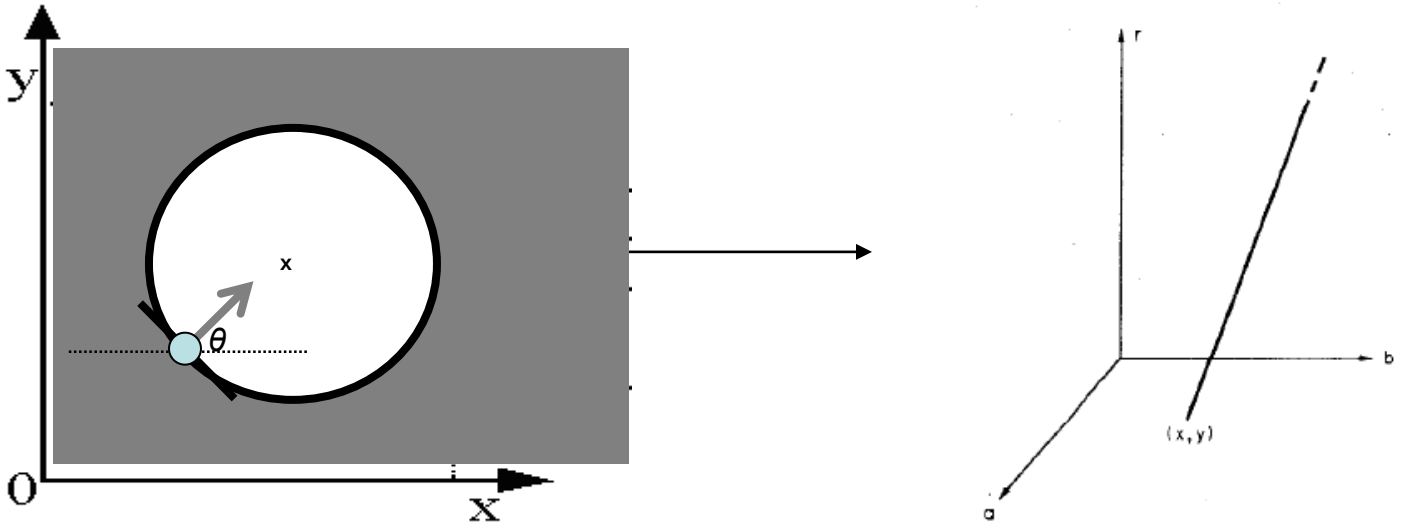


Image space

Hough space

Hough transform for circles

For each edge pixel (x,y) :

For each radius value r:

For each gradient direction θ:

// or use the estimated direction only

$a = x - r\cos(\theta)$

$b = y + r\sin(\theta)$

$H[a,b,r]$ += 1

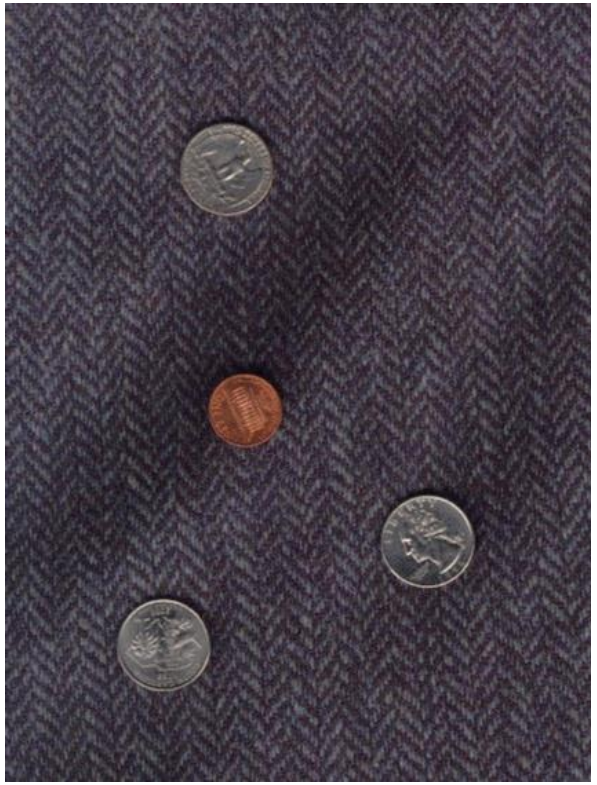
end for

end for

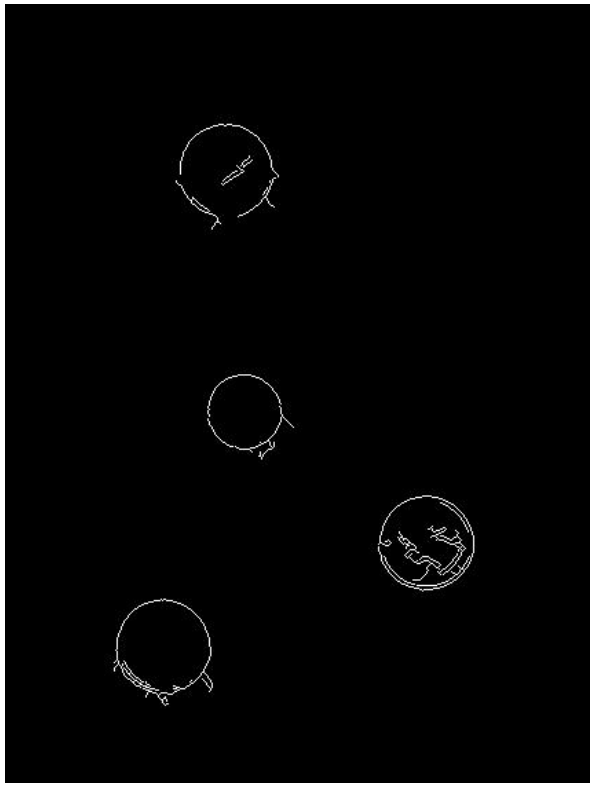
end for

Hough circle detection in action!

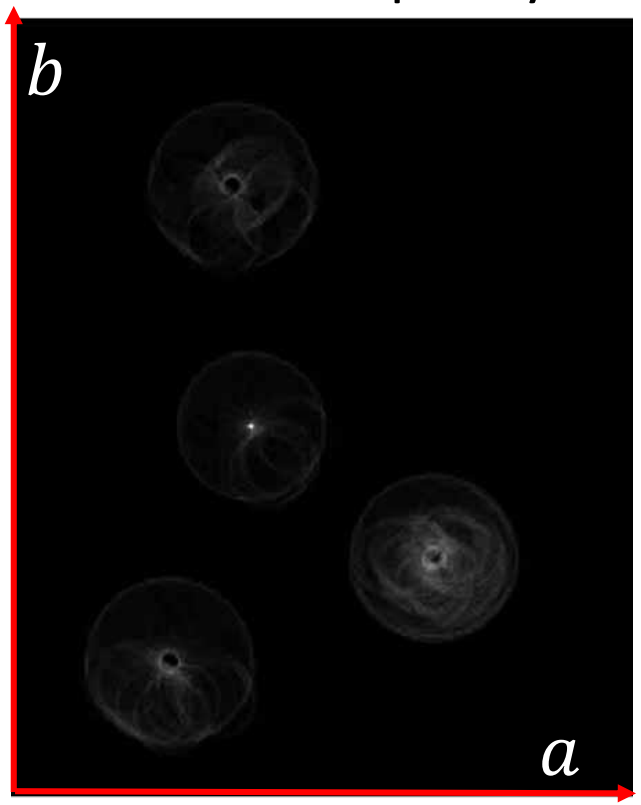
Original



Edges (e.g., Canny)



Votes for penny



$(x_i - a)^2 + (y_i - b)^2 = r^2$

Given a known radius:

$r = 50\ pixels,$

what are the center coordinates, i.e.,

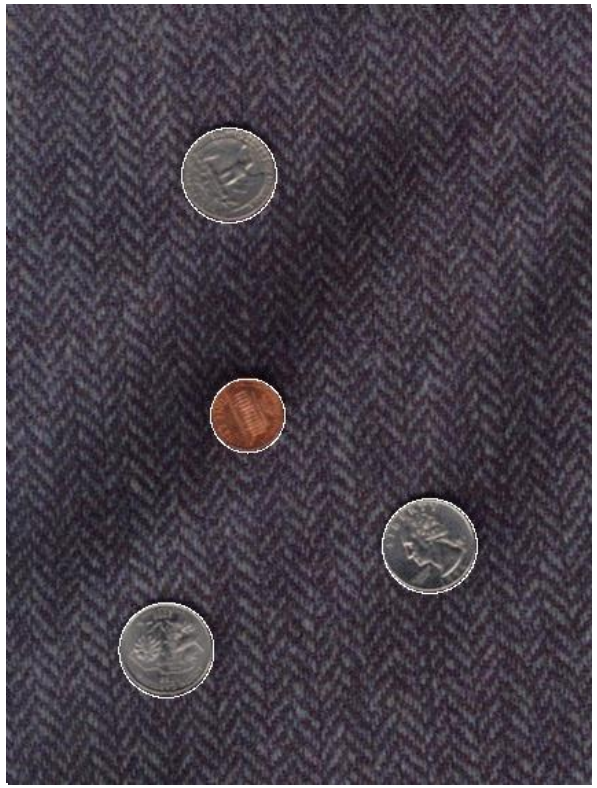
$a = ?\ pixels,$
$b = ?\ pixels\ .$

Comment: here we use a separate HT for each coin size.

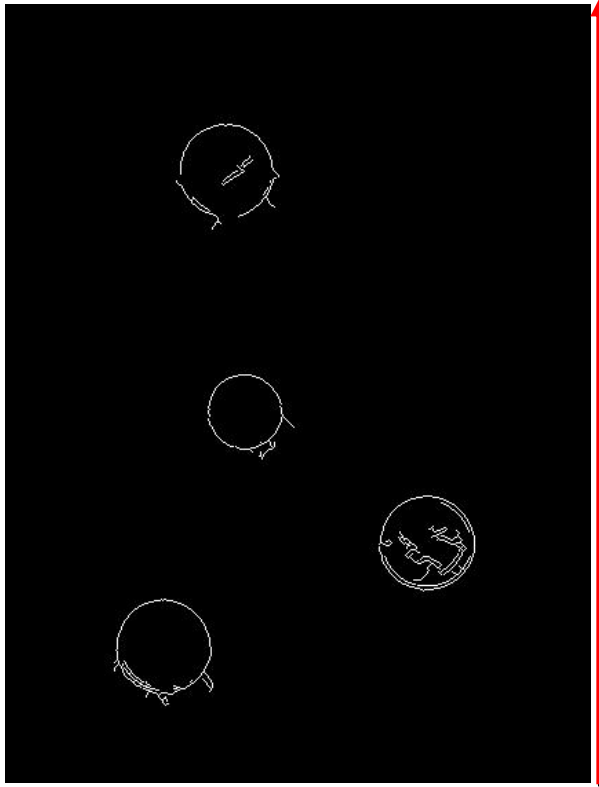
Images from: Vivek Kwatra

Hough circle detection in action!

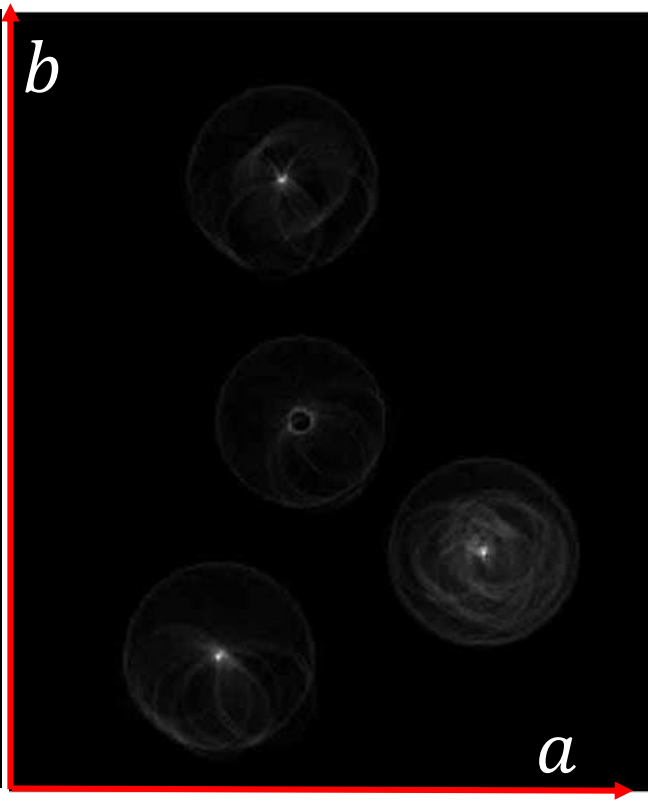
Original



Edges (e.g., Canny)



Votes for 25cent



Combined detections

$(x_i - a)^2 + (y_i - b)^2 = r^2$

Given a known radius:

$r = 50\ pixels,$

what are the center coordinates, i.e.,

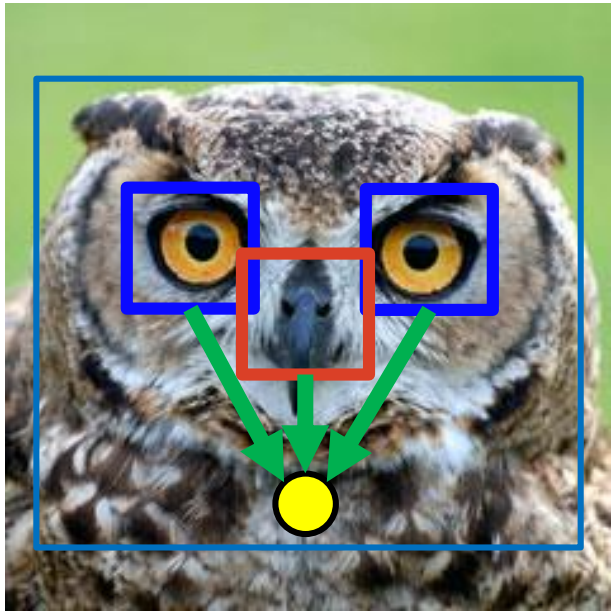
$a = ?\ pixels,$
$b = ?\ pixels\ .$

Comment: here we use a separate HT for each coin size.

Generalized Hough transform (GHT)

Building a model to detect objects by GHT – intuition:

- Assume we know how to detect parts (recognize+localize), i.e., eyes and beak of an howl. Task: *create an howl head detector.*
- Encode parts by displacements to the neck center.



The owl head model:

Given a part, where is the neck center?

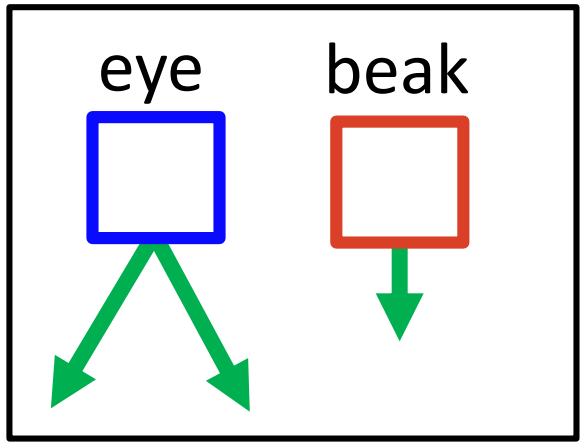
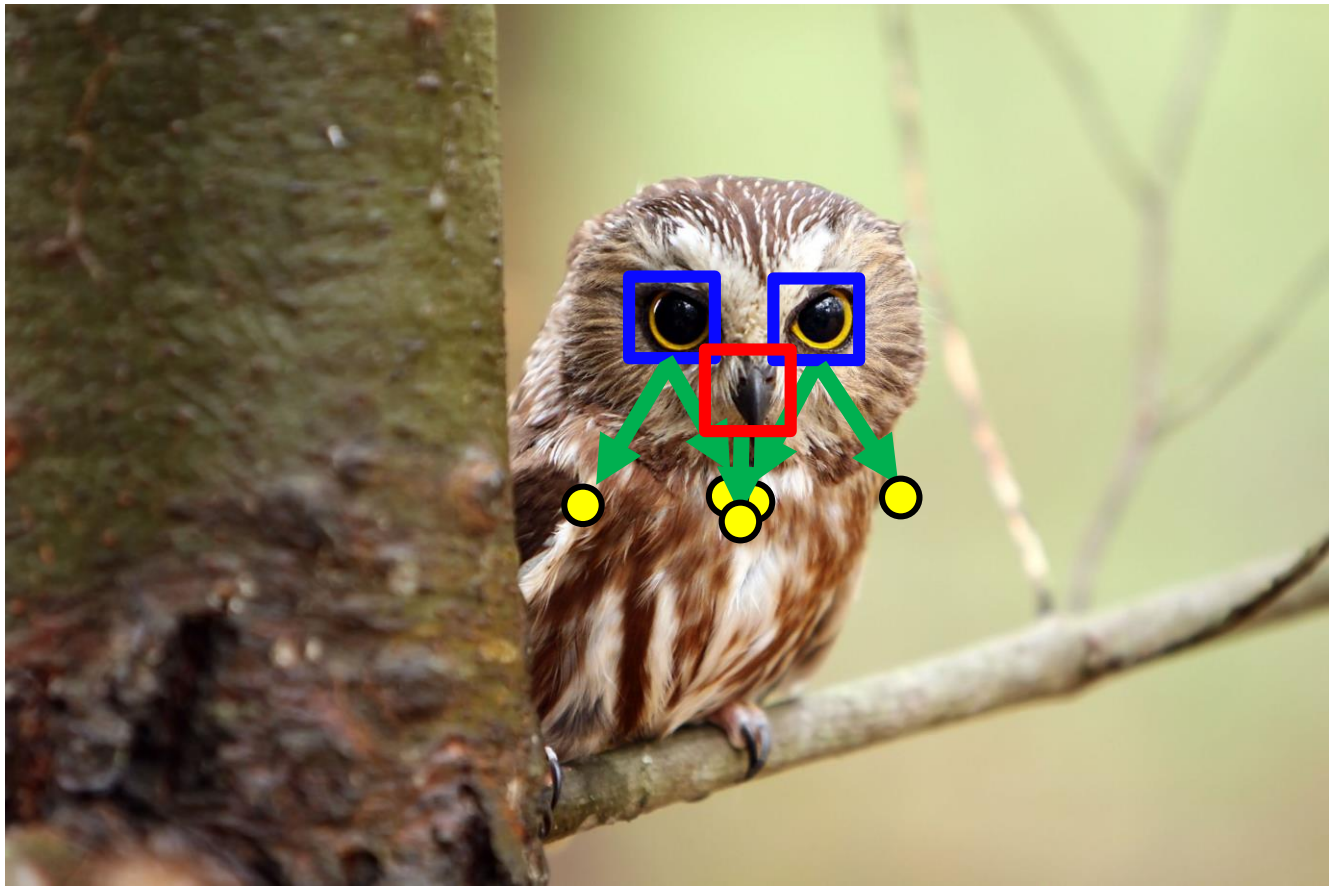


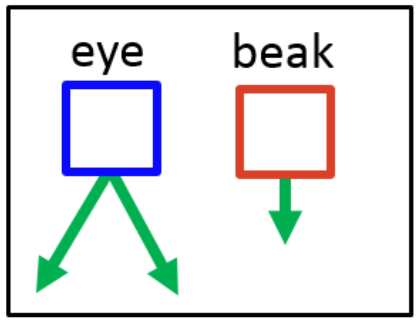
Image from: http://www.exposureguide.com/

Generalized Hough transform (GHT)

- Detection – intuition



The owl head model:



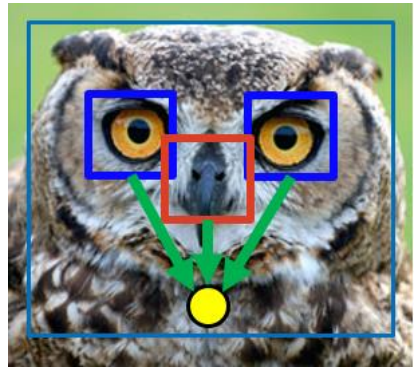
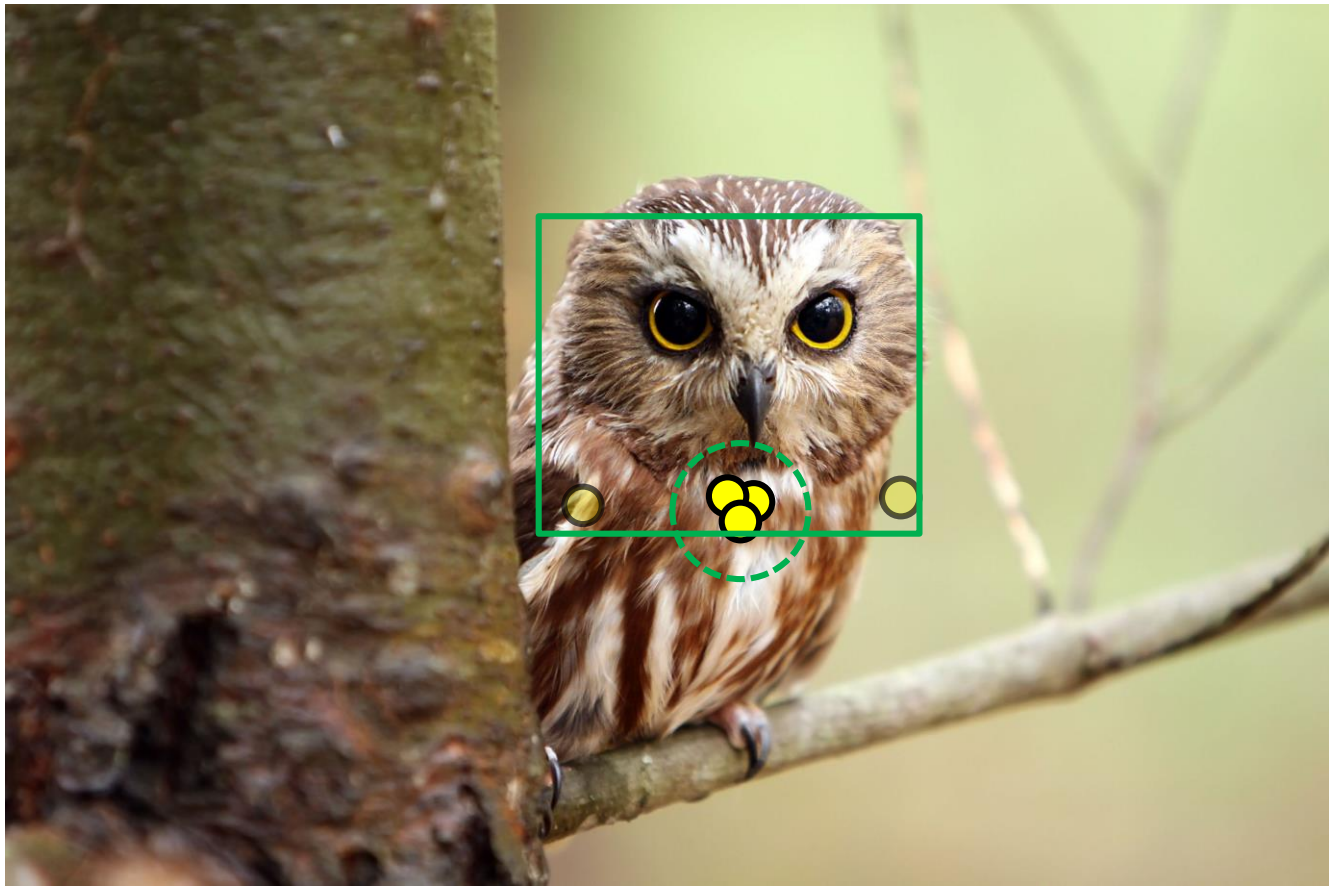


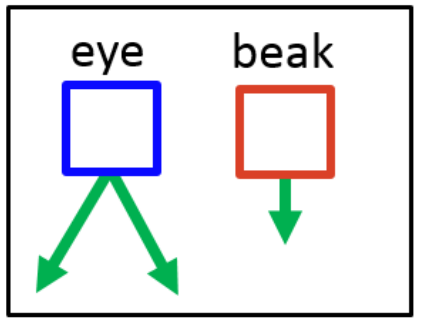
Image from: http://pugetsoundbirds.org/projects/owl-project/

Generalized Hough transform (GHT)

- Detection – intuition



The owl head model:



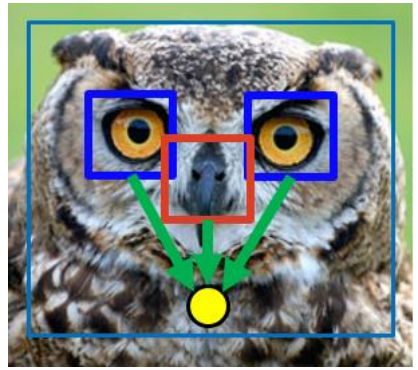
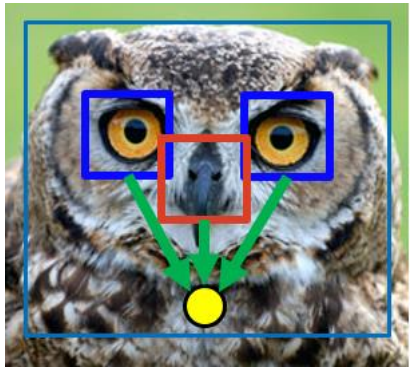
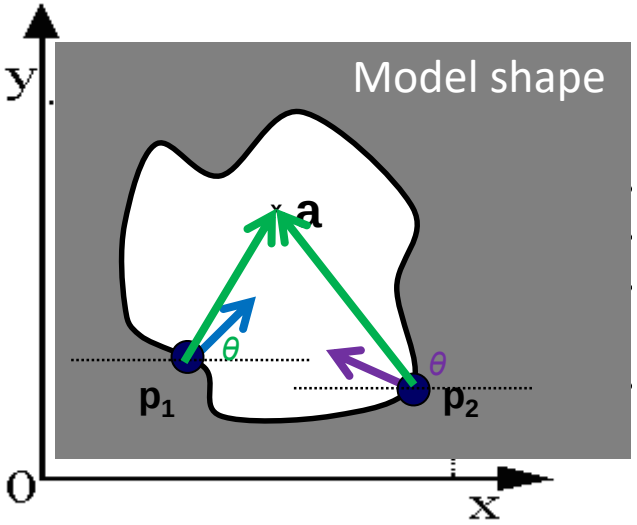


Image from: http://pugetsoundbirds.org/projects/owl-project/

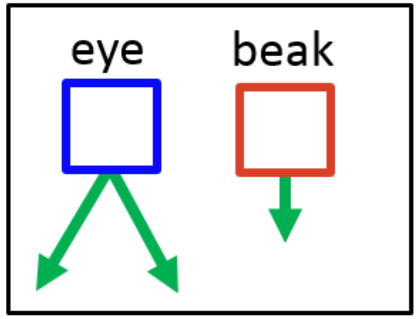
GHT for shape-based models

- Define the shape model by edge points and a reference point.





The owl head model:



Edge/gradient direction | Voting vector

θ	…
θ	…
⋮	

Model learning:

For each edge point calculate the displacement vector to the reference point:
$\mathbf{r = a - p_i}$.

Collect displacements in table, indexed by gradient direction θ.

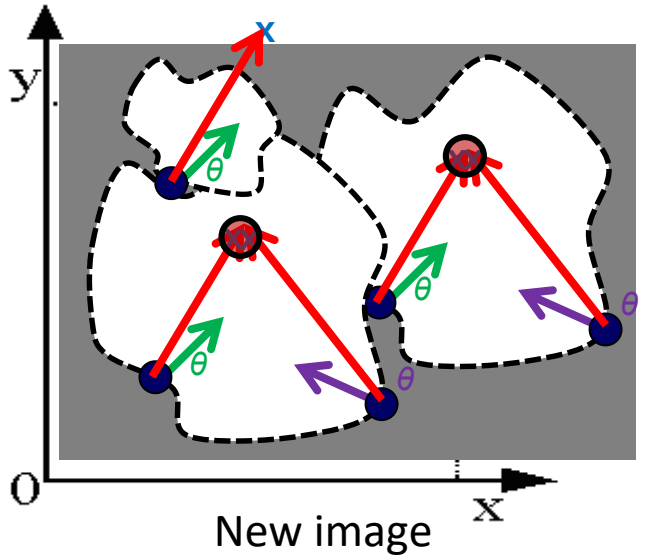
[Dana H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, 1980] Slide credit: Kristen Grauman

GHT for shape-based models

Detection:

For each edge point:

- Use its gradient orientation θ to index into the table.
- Use the displacement vectors **r** to cast a vote for the center.



New image

θ	…
θ	…
⋮	

Assumption: the only transformation is the translation (orientation+scaling are fixed)

Voting: Practical advices

- First minimize irrelevant responses
  (use only edges with significant magnitude of gradient)
- Appropriately discretize the parametric space
  - Too coarse: votes from different lines fall into the same accumulator
  - Too fine: losing lines – due to noise, collinear points cast votes into nearby (BUT DIFFERENT) accumulators.

- Vote for neighboring cells as well (smoothing accumulator by Gaussian filtering)
- Use the gradient direction to reduce dof by 1.

Slide credit: Kristen Grauman

Hough transform: +/-

Pros

- Each point is processed independently:
  - robustness to partial occlusion,
  - highly parallelizable.
- Robustness to noise: noise will unlikely contribute consistently to a single cell
- Can detect multiple instances of a single model in one pass.

Cons

- Time complexity increases exponentially with the number of free parameters.
- Spurious shapes may generate false local maxima in the parametric space.
- Quantization: Not particularly easy to choose a proper accumulator cell size – Application dependent!

References

- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition), (prva izdaja dostopna na spletu)
- R. Szeliski,Computer Vision: Algorithms and Applications, 2010
- R. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, 2nd Edition, Cambridge University Press, 2004
- Kristen Grauman, „Computer Vision“, lectures