COMP2005

Derivative Filters

In 1D

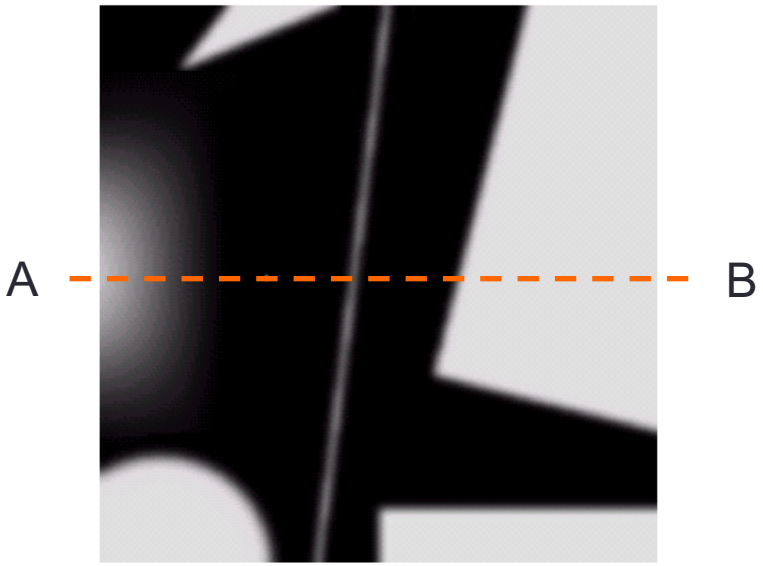
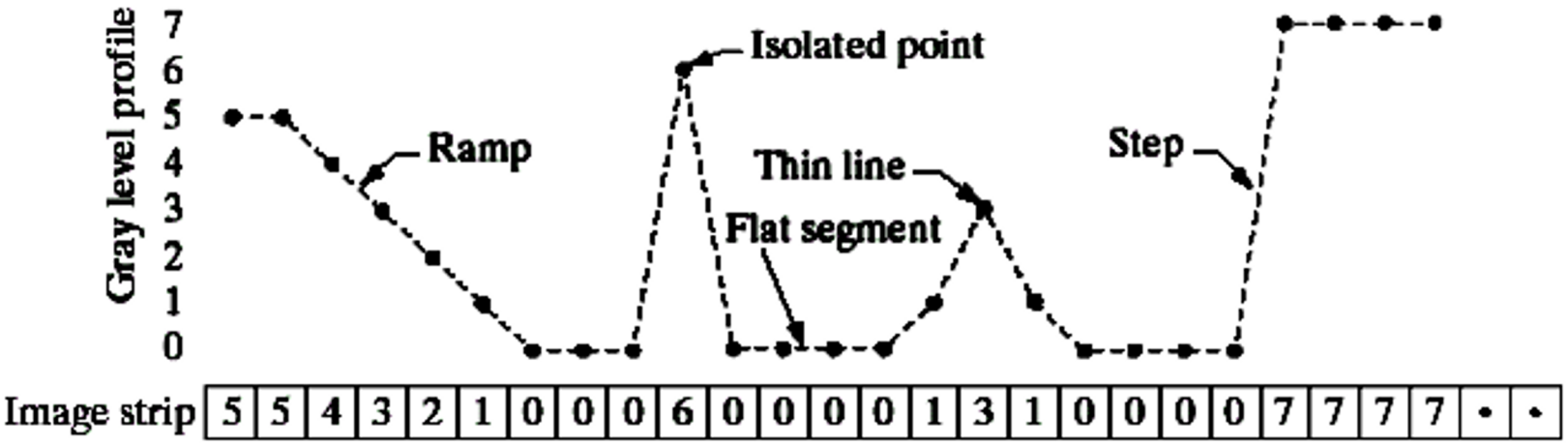


Image features are often characterised by changes in intensity

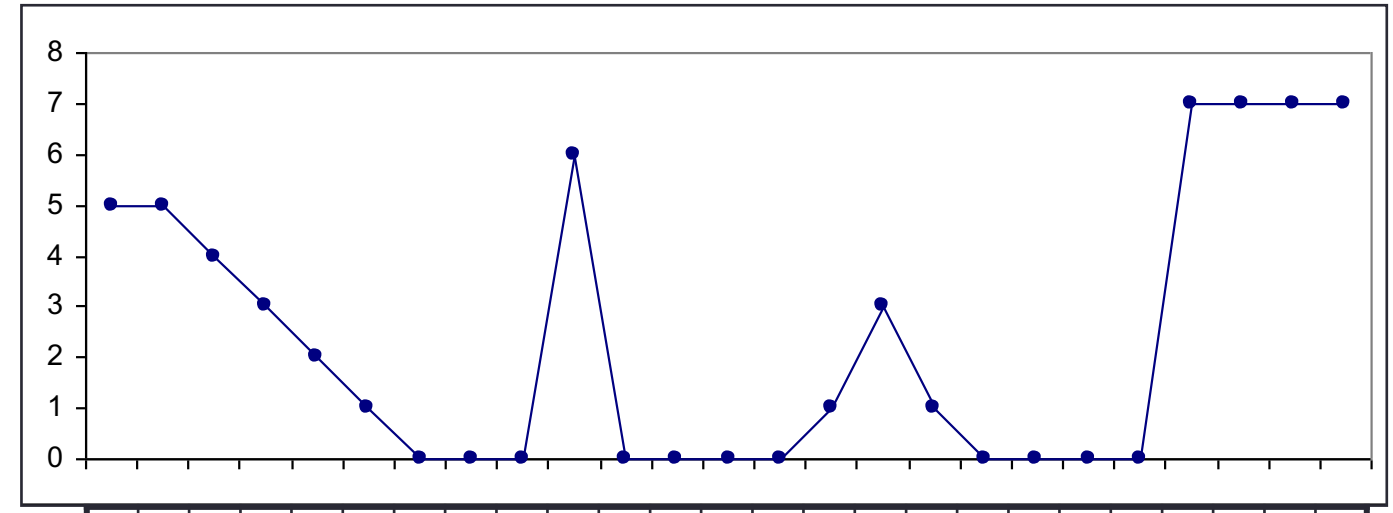


1st Derivative

- The 1st derivative of a function can be approximated by:

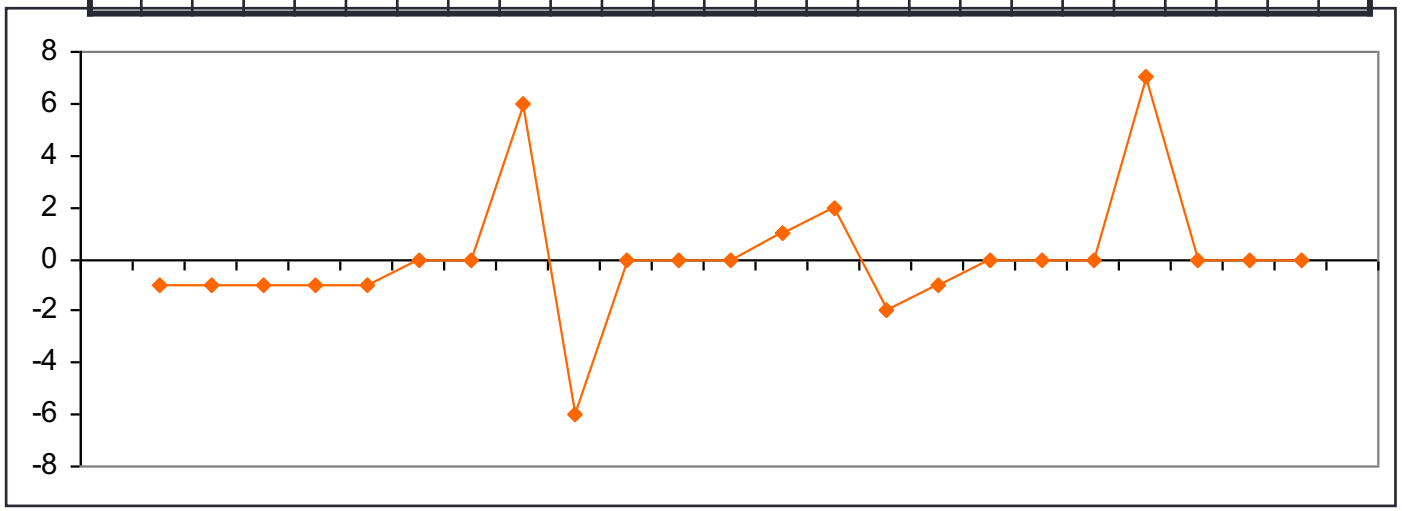
$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

- The difference between neighbouring values and measures the rate of change of the function
- Can also be approximated by (*f(x+1) – f(x-1))2,* etc.



5	5	4	3	2	1	0	0	0	6	0	0	0	0	1	3	1	0	0	0	0	7	7	7	7

	0	-1	-1	-1	-1	-1	0	0	6	-6	0	0	0	1	2	-2	-1	0	0	0	7	0	0	0



Raw data

1st derivative

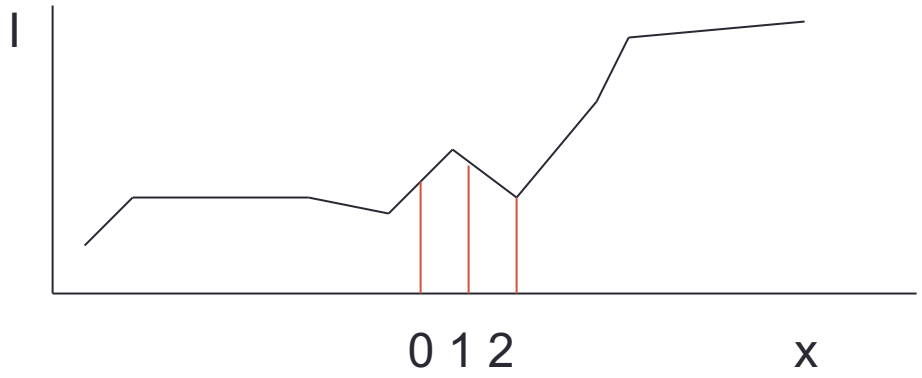
2nd Derivative

- The formula for the 2nd derivative of a function is:

$$\frac{\partial^2 f}{\partial^2 x} = f(x+1) + f(x-1) - 2f(x)$$

- Simply takes into account the values both before and after the current value
- Derived by estimating the 1st derivative at x + 0.5 and x – 0.5 and computing the derivative of the resulting data

2nd Derivative

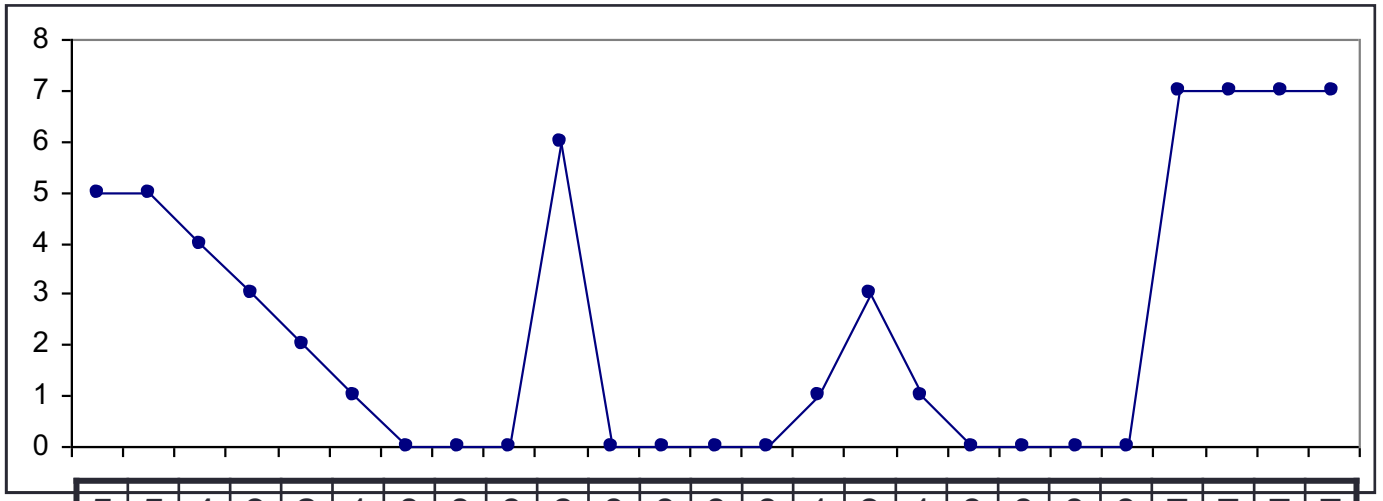


$I''(1) = (I'(1.5) - I'(0.5))/1$

$I'(0.5) = (I(1) - I(0))/1$ and $I'(1.5) = (I(2) - I(1))/1$

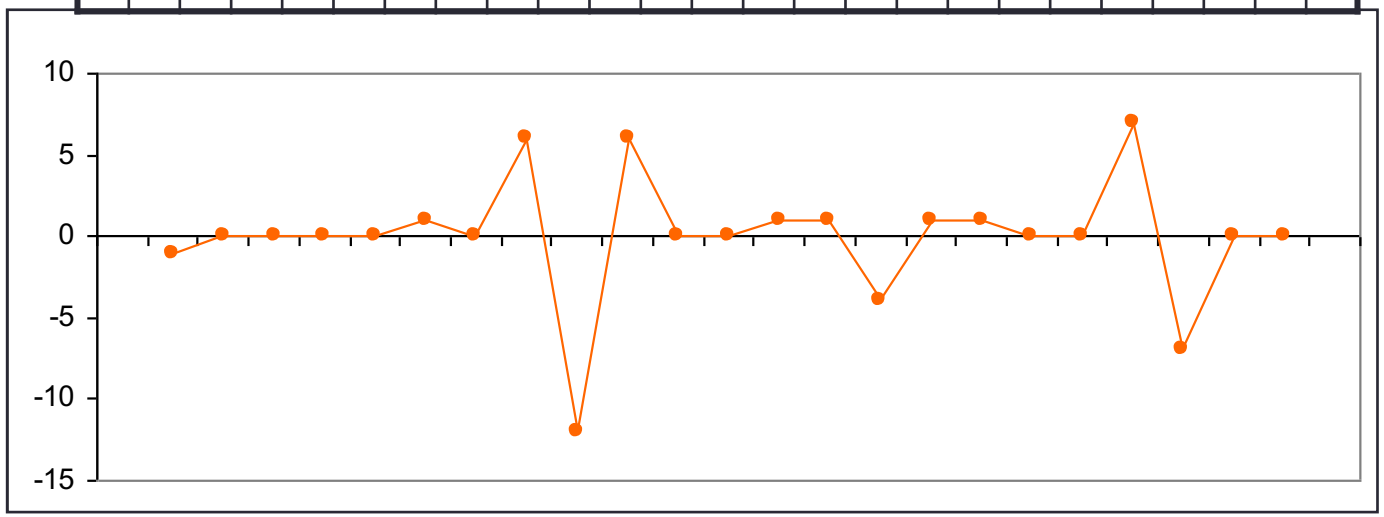
$\therefore\ I''(I) = 1.I(0) - 2.I(1) + 1.I(2)$

1	-2	1



5	5	4	3	2	1	0	0	0	6	0	0	0	0	1	3	1	0	0	0	0	7	7	7	7

	-1	0	0	0	0	1	0	6	-12	6	0	0	1	1	-4	1	1	0	0	7	-7	0	0	



Raw data

2nd derivative

Derivatives in 2D

- 2nd derivatives generalize to 2D quite easily, implementing a 1st derivative in 2D is a little more complex
- For a function *f(x, y)* the gradient of *f* at coordinates *(x, y)* is given as the column vector:

$$\nabla \mathrm{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

- Computation of the 1st derivative can't be done by convolution alone

1st Derivative Filtering

- The magnitude of the 1st derivative vector is

$$\nabla f = mag(\nabla \mathrm{f})$$
$$= \left[G_x^2 + G_y^2\right]^{1/2}$$
$$= \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2}$$

which can be simplified to

$$\nabla f \approx |G_x| + |G_y|$$

1st Derivative Filters

- Many1st derivative filters have been proposed
- Roberts' Cross Operators

1	0
0	-1

0	1
-1	0

- Sobel Operators

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

G_x

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

G_y

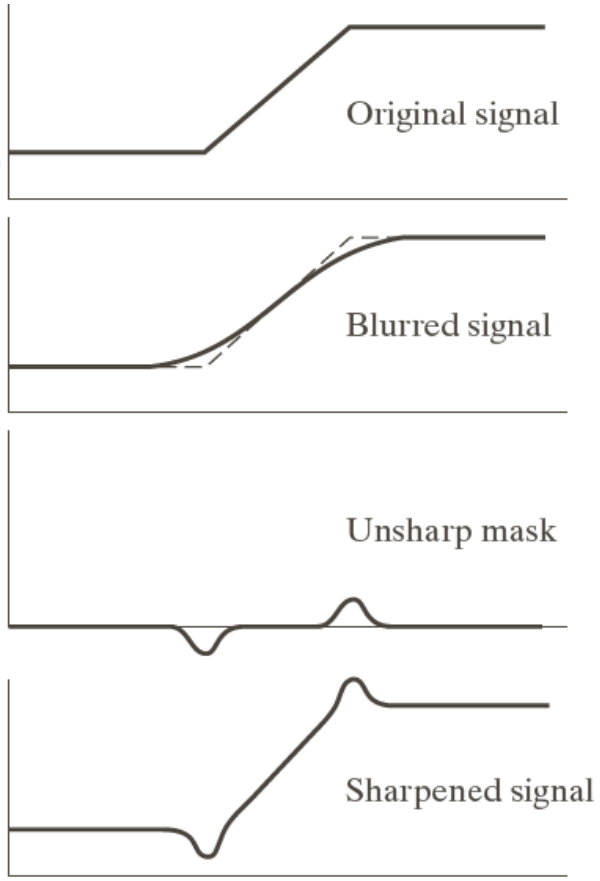
These operators are most commonly associated with edge detection

COMP2005

Image Sharpening

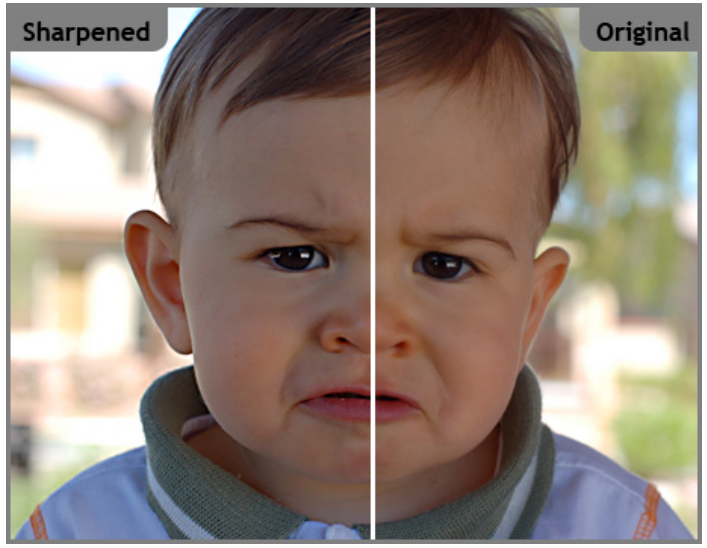
Edge Enhancement: Unsharp Masking

- Edges are important
- Sometimes we want to enhance them without (much) affecting the rest of the image
  - Take the original image
  - Gaussian smooth it
  - Subtract the smoothed version from the original to make an *unsharp mask*
  - Add the mask to the original to make the edge appear more obvious



Unsharp Masking

- Makes edges noticeably sharper
  - Even if they are noise
  - Sometimes too much



Derivative Filters

- Unsharp filtering enhances edges by comparing the original with a smoothed image
  - relies on the smoothing effect of a Gaussian function introducing a difference between original and processed images
  - parameterised by σ
  - simple, but effect is hard to predict, so hard to parameterise
- A more direct way to highlight edges and other features associated with high image gradients is to estimate derivatives…….

Image Sharpening with Derivatives

- The 2nd derivative is more useful for image enhancement than the 1st derivative
  - Stronger response to fine detail
  - Simpler implementation
- The most common sharpening filter is the *Laplacian*
  - Isotropic
  - One of the simplest sharpening filters
  - Straightforward digital implementation via convolution

The Laplacian

$$\nabla^2 f = \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y}$$

$$\frac{\partial^2 f}{\partial^2 x} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial^2 y} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

The Laplacian

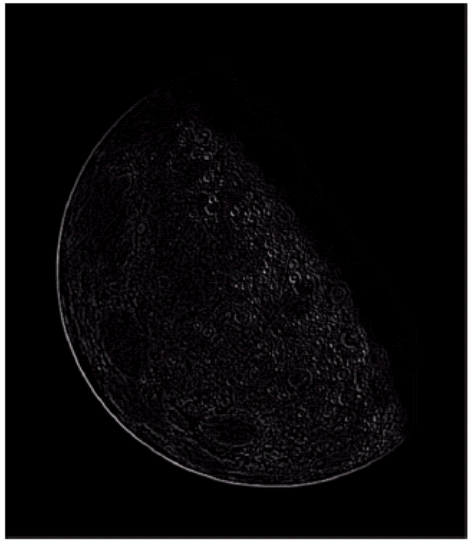
$$\nabla^2 f = [f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)] - 4f(x, y)$$

0	1	0
1	-4	1
0	1	0

The Laplacian

- Highlights edges and other discontinuities

Original Image



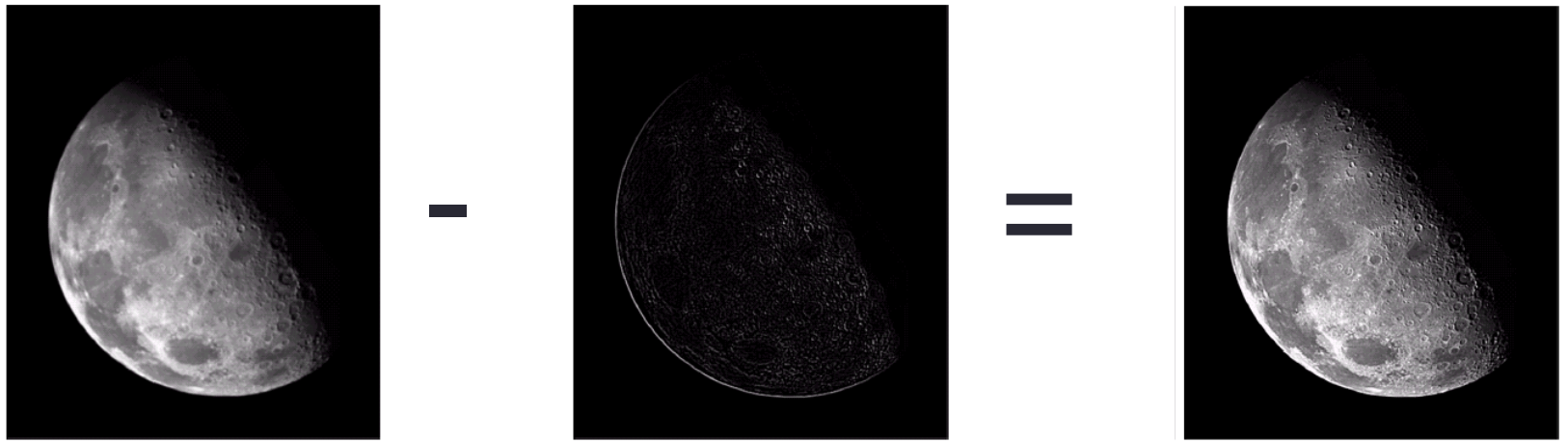
Laplacian Filtered Image

Laplacian Filtered Image Scaled for Display

Could do better?

- The result of Laplacian filtering is not an enhanced image
  - subtract the Laplacian result from the original image to generate the final sharpened enhanced image, c.f. unsharp masking

$$g(x,y) = f(x,y) - \nabla^2 f$$



Original Laplacian Filtered Sharpened

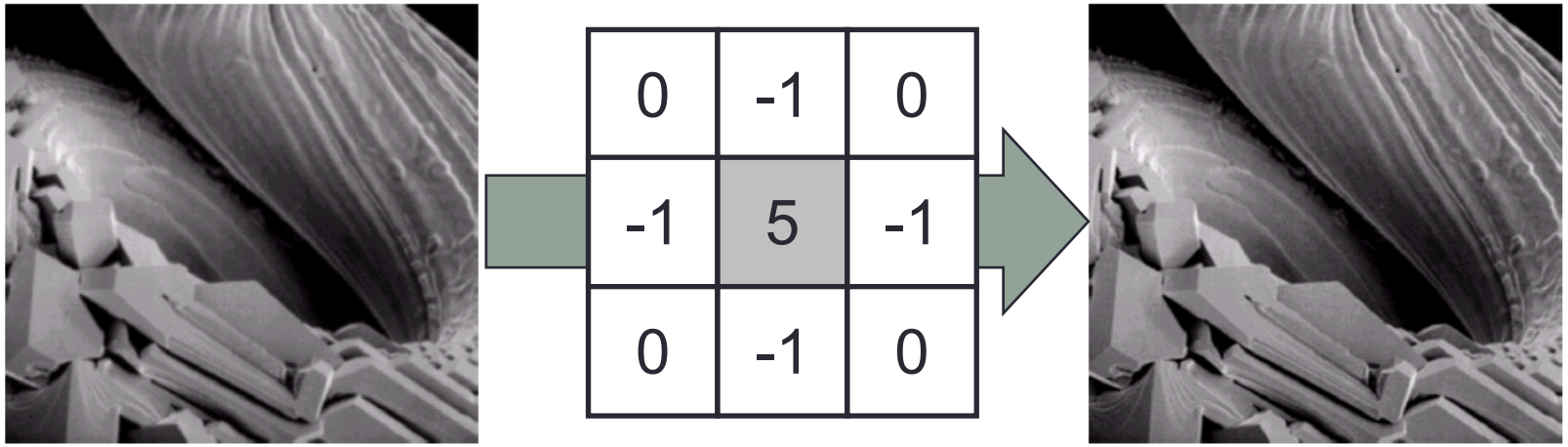
Laplacian Enhancement

A Single Enhancement Operator

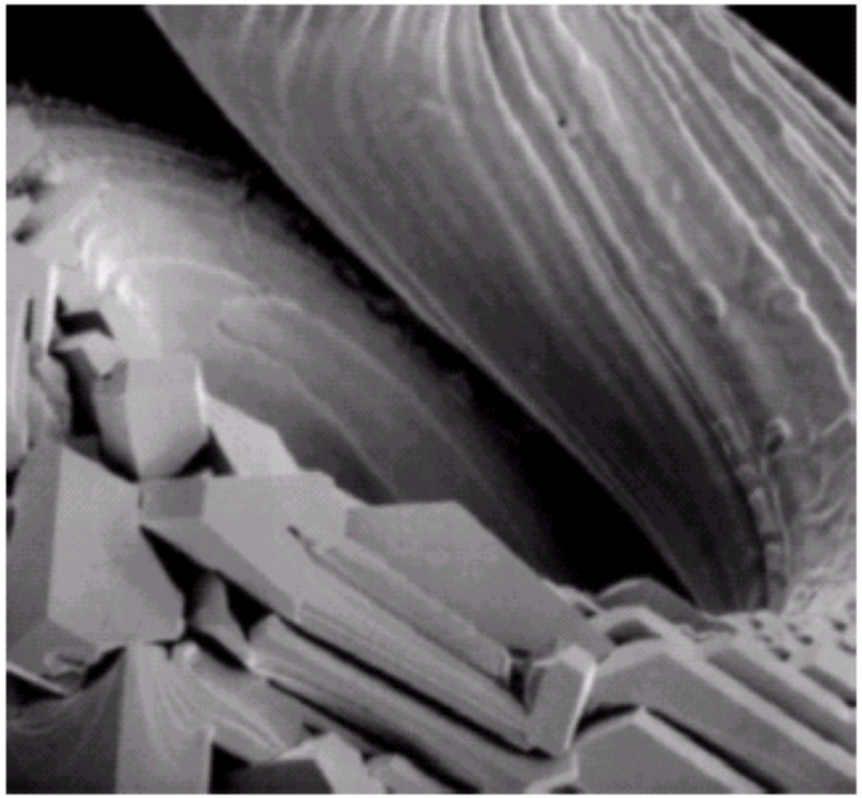
$$
\begin{aligned}
g(x,y) &= f(x,y) - \nabla^2 f \\
&= f(x,y) - [f(x+1,y) + f(x-1,y) \\
&\qquad + f(x,y+1) + f(x,y-1) \\
&\qquad - 4f(x,y)] \\
&= 5f(x,y) - f(x+1,y) - f(x-1,y) \\
&\qquad - f(x,y+1) - f(x,y-1)
\end{aligned}
$$

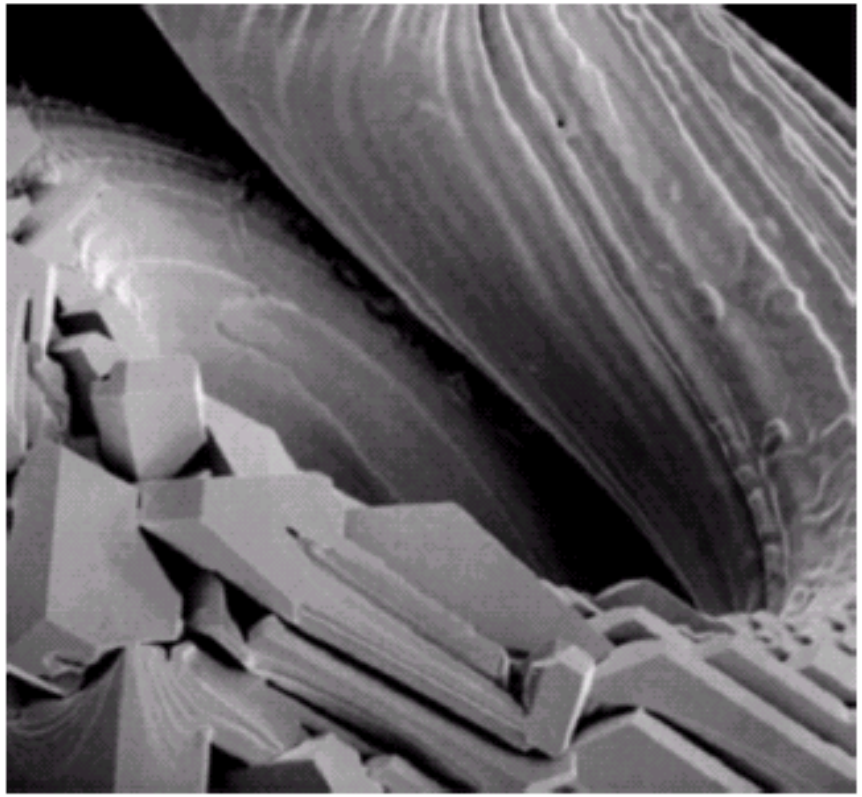
A Single Operator

- Convolution with this operator performs image sharpening in a single step



A Single Operator





Variations on the Theme

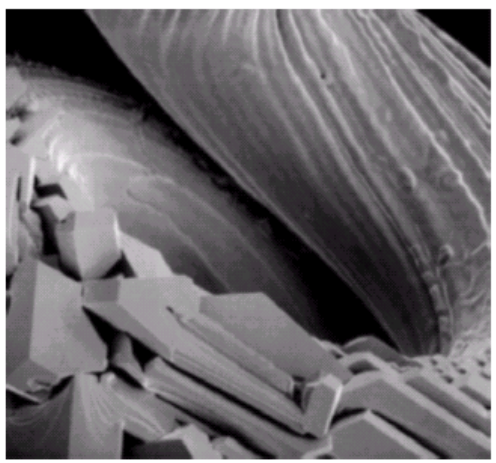
- Some formulations take 2nd derivatives measured across the diagonals into account

0	1	0
1	-4	1
0	1	0

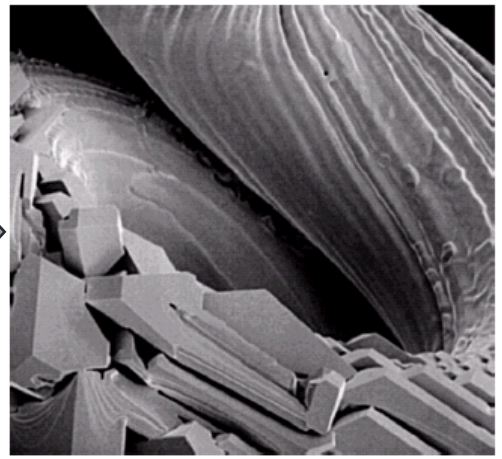
Basic

1	1	1
1	-8	1
1	1	1

Extended



-1	-1	-1
-1	9	-1
-1	-1	-1

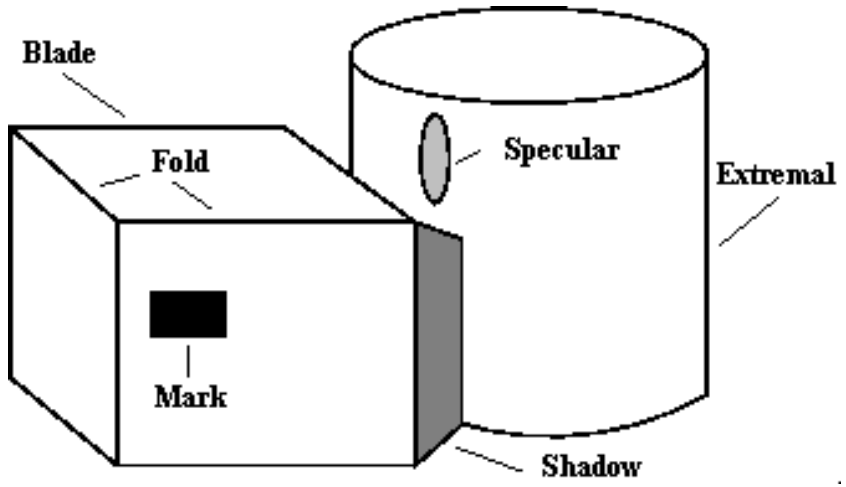


COMP2005

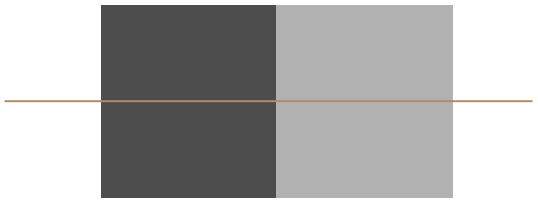
What is Edge Detection?

Edge Detection

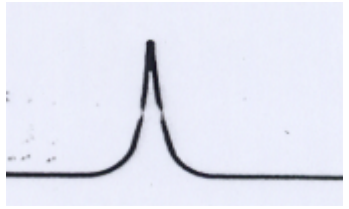
- First step in many image analysis and computer vision processes and applications
- The goal is to mark points at which image intensity changes sharply – edges
- Sharp changes in image properties reflect events/changes in the world
- This is only an assumption,
  but it is usually true



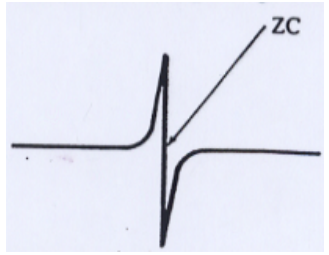
The Theory



- An idealised image of an edge



- 1st derivative: a peak



- 2nd derivative: a zero-crossing

To detect edges find peaks in the 1st derivative of intensity or zero-crossings in the 2nd derivative

The Result

```
>> im = imread('cameraman.tif');
>> edges = edge(im, 'Canny');
>> imshowpair(im, edges, 'montage');
```

COMP2005

Edge Detection using 1st Derivative Filters

1st Derivative Filters

- Roberts' Cross Operators

1	0
0	-1

0	1
-1	0

- Sobel Operators

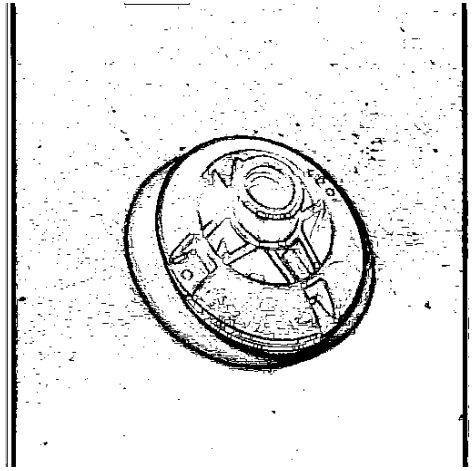
-1	0	1
-2	0	2
-1	0	1

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

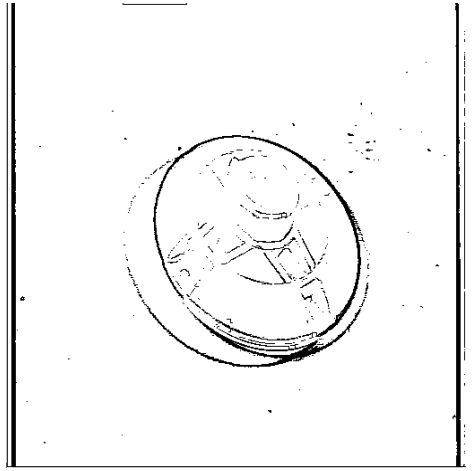
- Applied separately and results combined to estimate magnitude

Detection & Thresholding

- Significant peaks in magnitude of 1st derivative are high
- Apply a threshold, all peaks higher than the threshold value are significant, all others are ignored



- Too low



- Too high

Edge Magnitude & Direction

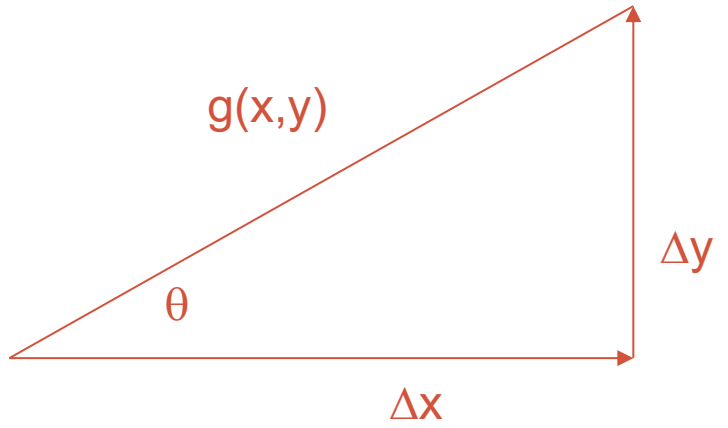
- For an image function, I(x,y), the gradient magnitude, g(x,y) is given by

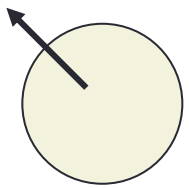
$$g(x,y) \cong (\Delta x^2 + \Delta y^2)^{1/2}$$

- The gradient direction, $\theta(x,y)$, gives the direction of steepest image gradient

$$\theta(x,y) \cong \text{atan}(\Delta y/\Delta x)$$

- This gives the direction of a line perpendicular to the edge

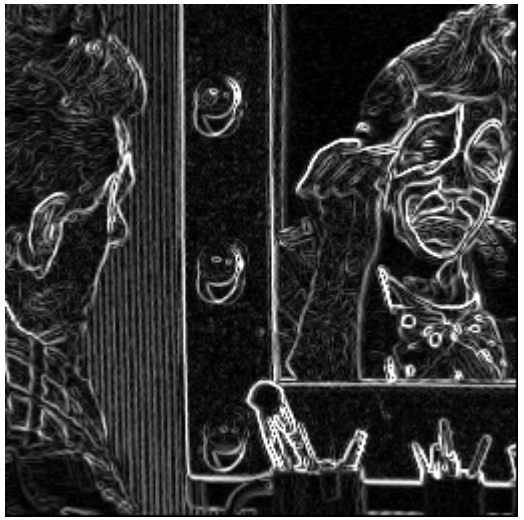




Roberts' Cross Operator

- Very quick to compute - 4 pixels, only subtractions and additions, but is very sensitive to noise and only gives a strong response to very sharp edges

Original



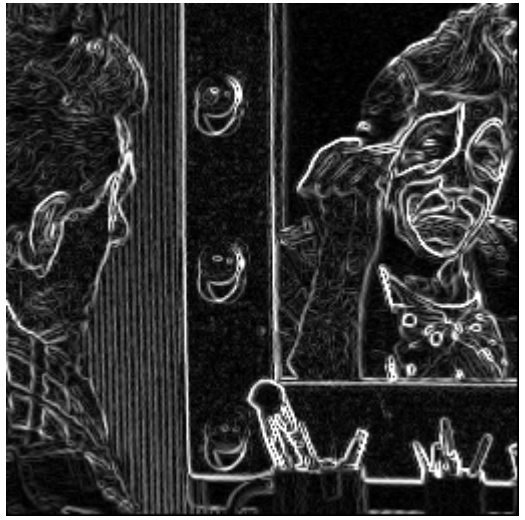
Cross Operator

Thresholded

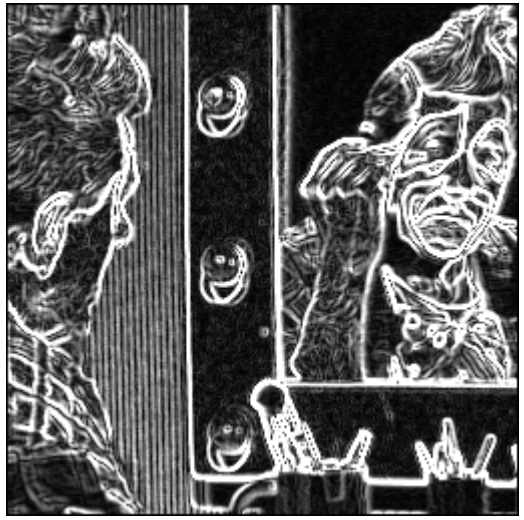
Sobel vs Roberts

- Both use a user-supplied threshold. Sobel is still in use, Roberts is less common.
- Larger Sobel operators are more stable in noise

Original



Roberts

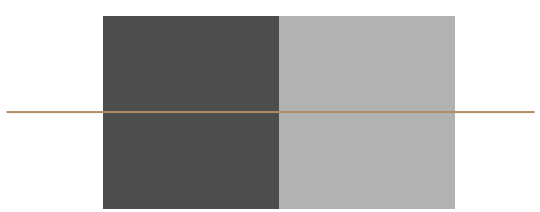


Sobel

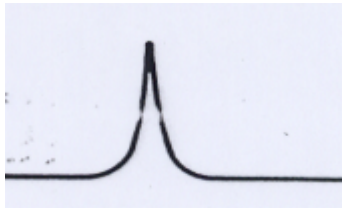
COMP2005

Edge Detection using 2nd Derivatives

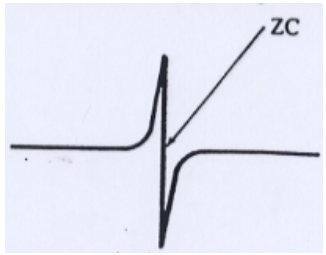
The Theory



- An idealised image of an edge



- 1st derivative: a peak



- 2nd derivative: a zero-crossing

To detect edges find peaks in the 1st derivative of intensity or zero-crossings in the 2nd derivative

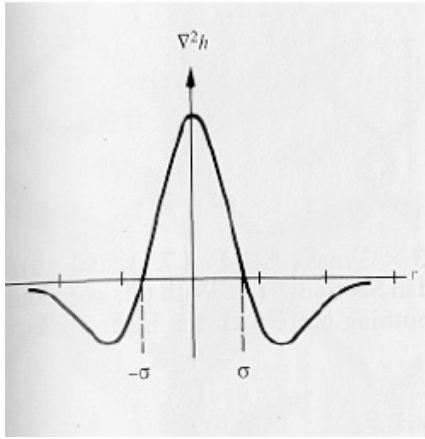
2nd Derivatives: Marr-Hildreth

- Biologically inspired
- Gaussian smooth, compute Laplacian

OR

- Convolve with the Laplacian of a Gaussian

$$\nabla^2[f(x,y) * G(x,y)] = \nabla^2 G(x,y) * f(x,y)$$



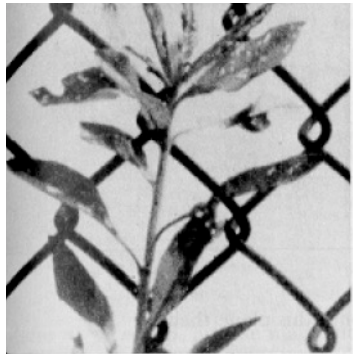
Laplacian of Gaussian (LoG)

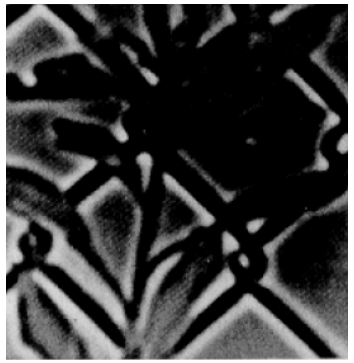
5 × 5 Laplacian of Gaussian mask

0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

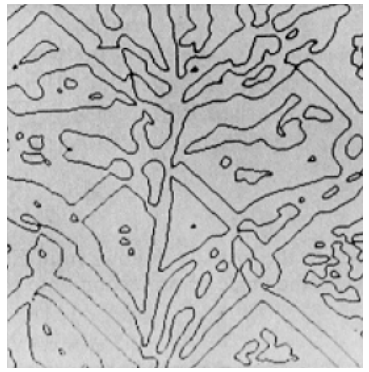
17 × 17 Laplacian of Gaussian mask

0	0	0	0	0	0	−1	−1	−1	−1	−1	0	0	0	0	0	0
0	0	0	0	−1	−1	−1	−1	−1	−1	−1	−1	−1	0	0	0	0
0	0	−1	−1	−1	−2	−3	−3	−3	−3	−3	−2	−1	−1	−1	0	0
0	0	−1	−1	−2	−3	−3	−3	−3	−3	−3	−3	−2	−1	−1	0	0
0	−1	−1	−2	−3	−3	−3	−2	−3	−2	−3	−3	−3	−2	−1	−1	0
0	−1	−2	−3	−3	−3	0	2	4	2	0	−3	−3	−3	−2	−1	0
−1	−1	−3	−3	−3	0	4	10	12	10	4	0	−3	−3	−3	−1	−1
−1	−1	−3	−3	−2	2	10	18	21	18	10	2	−2	−3	−3	−1	−1
−1	−1	−3	−3	−3	4	12	21	24	21	12	4	−3	−3	−3	−1	−1
−1	−1	−3	−3	−2	2	10	18	21	18	10	2	−2	−3	−3	−1	−1
−1	−1	−3	−3	−3	0	4	10	12	10	4	0	−3	−3	−3	−1	−1
0	−1	−2	−3	−3	−3	0	2	4	2	0	−3	−3	−3	−2	−1	0
0	−1	−1	−2	−3	−3	−3	−2	−3	−2	−3	−3	−3	−2	−1	−1	0
0	0	−1	−1	−2	−3	−3	−3	−3	−3	−3	−3	−2	−1	−1	0	0
0	0	−1	−1	−1	−2	−3	−3	−3	−3	−3	−2	−1	−1	−1	0	0
0	0	0	0	−1	−1	−1	−1	−1	−1	−1	−1	−1	0	0	0	0
0	0	0	0	0	0	−1	−1	−1	−1	−1	0	0	0	0	0	0





filtering



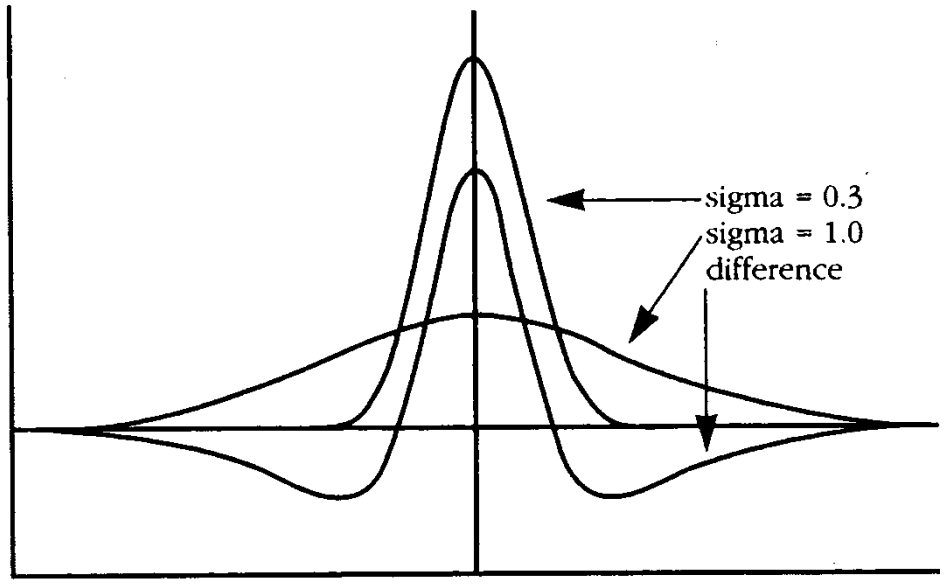
zero-crossings

Difference of Gaussians

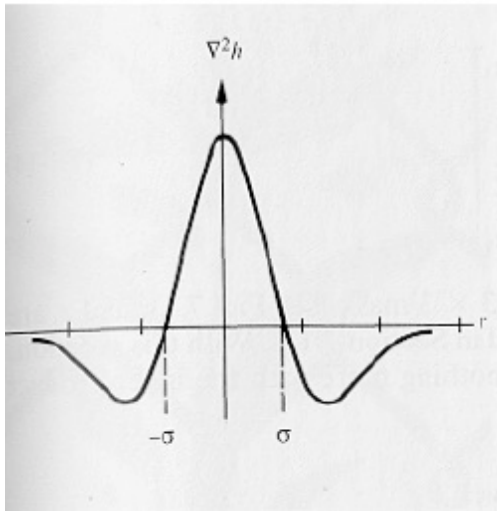
- The Laplacian of Gaussian can be approximated by the difference between two Gaussian functions:

$$\nabla^2 G \approx G(x, y; \sigma_1) - G(x, y; \sigma_2)$$

approximation



actual LoG



DoG Filtering

$$\nabla^2 G \approx G(x, y; \sigma_1) - G(x, y; \sigma_2)$$

$\sigma = 1$

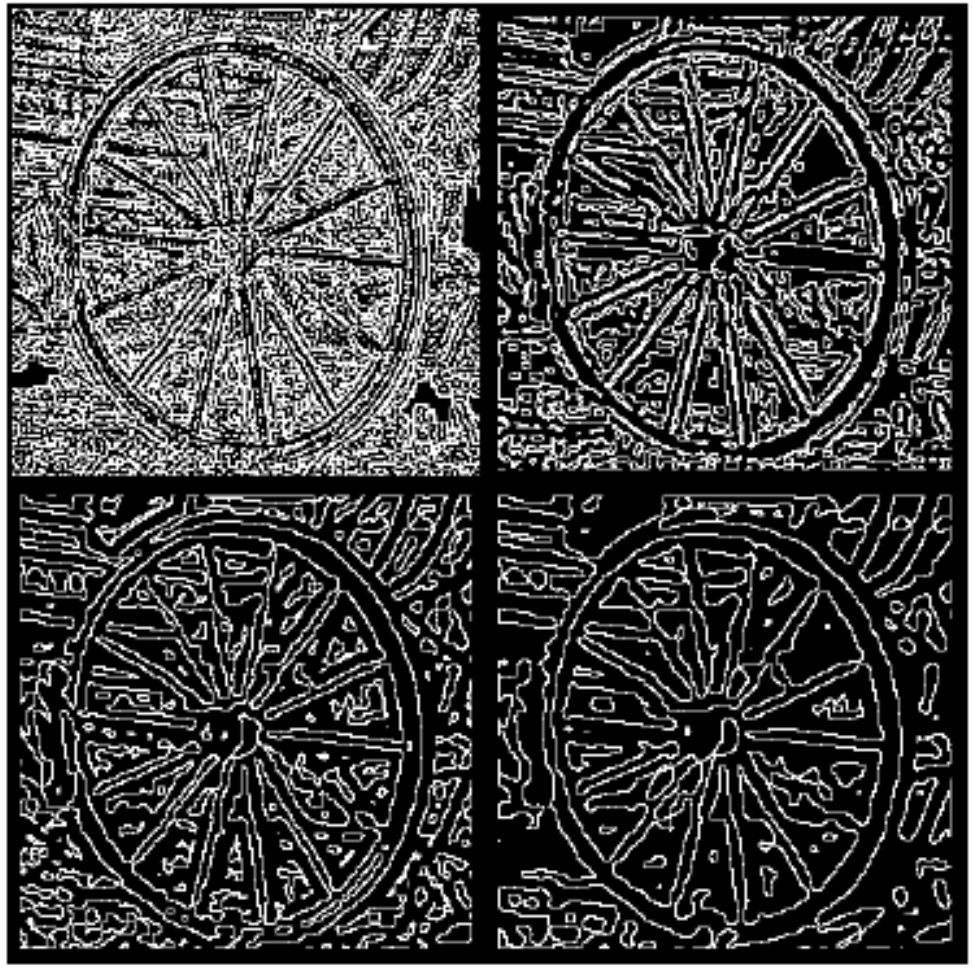
$\sigma = 2$

(b)-(a)

Ratio (σ_1/σ_2) for best approximation is about 1.6.
(Some people like $\sqrt{2}$.)

Marr-Hildreth

- Choice of σ gives flexibility



$\sigma = 1$	$\sigma = 2$
$\sigma = 3$	$\sigma = 4$

1st vs 2nd Derivative Methods

- Peaks in 1st derivative
  - strong response at edges, but also respond to noise
  - Peak detection and threshold selection need care
- Zero crossings in 2nd derivative
  - well-defined, easy to detect
  - must form smooth, connected contours
  - tend to round off corners
- 1st derivative methods are much more common in practical applications,
  - in part because of John Canny

COMP2005

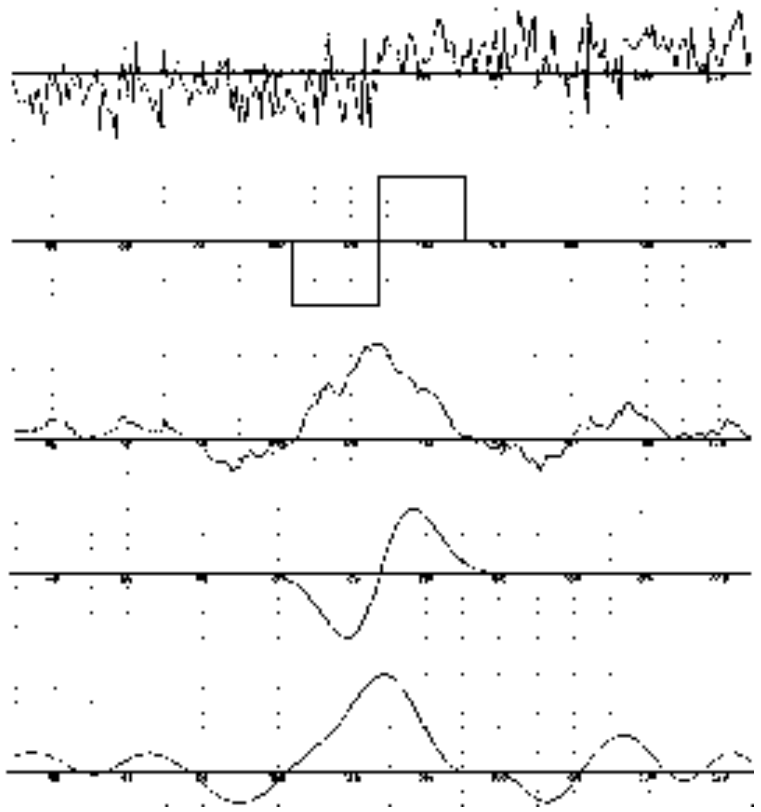
The Canny Operator

What Canny Did

- John Canny tried to find the optimal edge detector, assuming a perfect step edge in Gaussian noise
- Optimal means
  - Good Detection - it should mark all the edges and only all the edges
  - Good Localisation - the points marked should be as close to the real edge as possible
  - Minimal Response - each edge should be reported only once
- Canny used the *Calculus of Variations*: finds the function which best satisfies some functional

The Canny Operator

- The optimal detector was a sum of 4 exponential terms, but is very closely approximated by the 1st derivative of a Gaussian
  - i.e. 1st derivative of a Gaussian smoothed image
- Gives a cleaner response to a noisy edge than square operators
- Most implementations are 2D Gaussian smoothing + Roberts style derivative



Roberts

Canny

Non-Maximal Suppression

- The Canny operator's response is cleaner than Sobel or Roberts, but it needs an explicit step to enforce Minimal Response

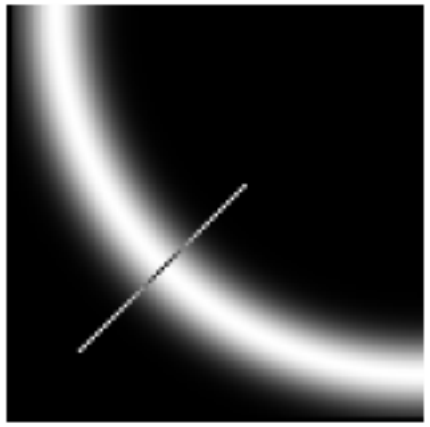
Canny Operator

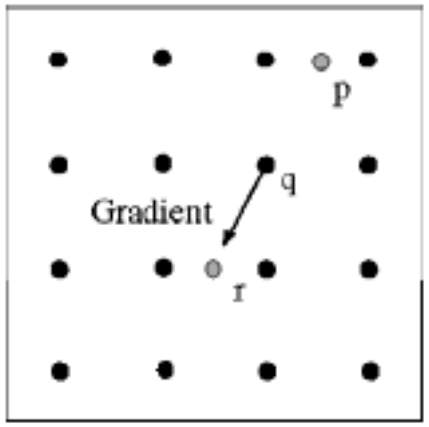
Non-Maximal Suppression

- Thresholding raw operator response would leave thick lines

Non-Maximal Suppression

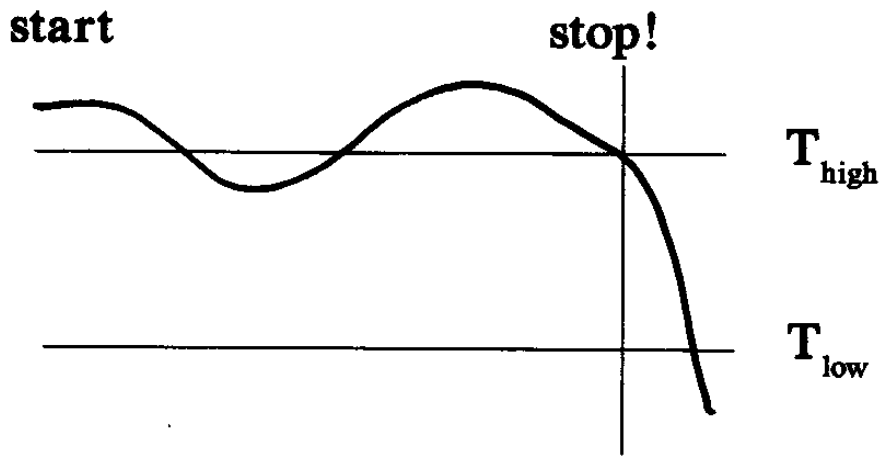
- Check if pixel is a local maximum along the gradient direction
- Select a single maximum across the width of the edge





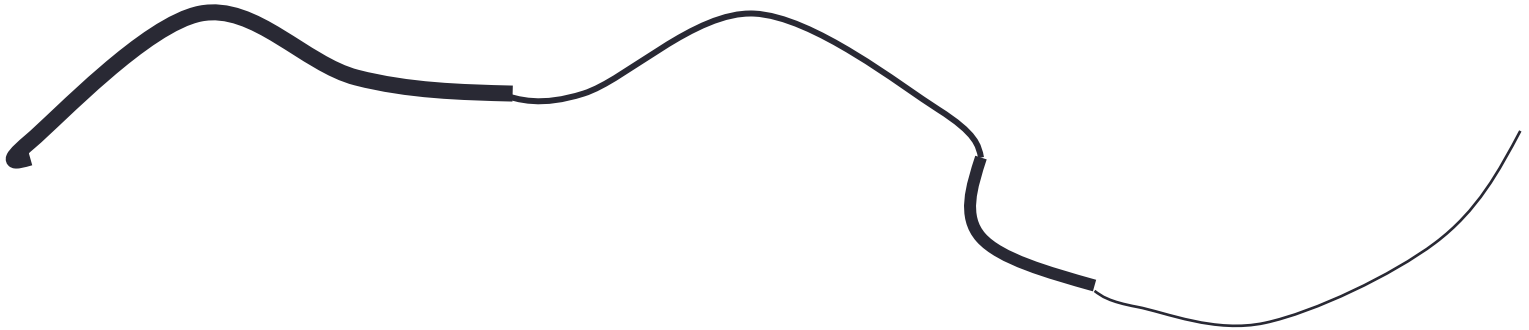
Thresholding with Hysteresis

- Simple thresholding tests each pixel independently: edges aren't really independent, they make up lines
- The industry standard edge thresholding method
  - Allows a band of variation, but assumes continuous edges
  - User still selects parameters, but its easier, less precise



Thresholding with Hysteresis

- The effect is to keep weak edges if they connect strong edges, as long as
  - the strong edges are really strong
  - the weak edges aren't really weak

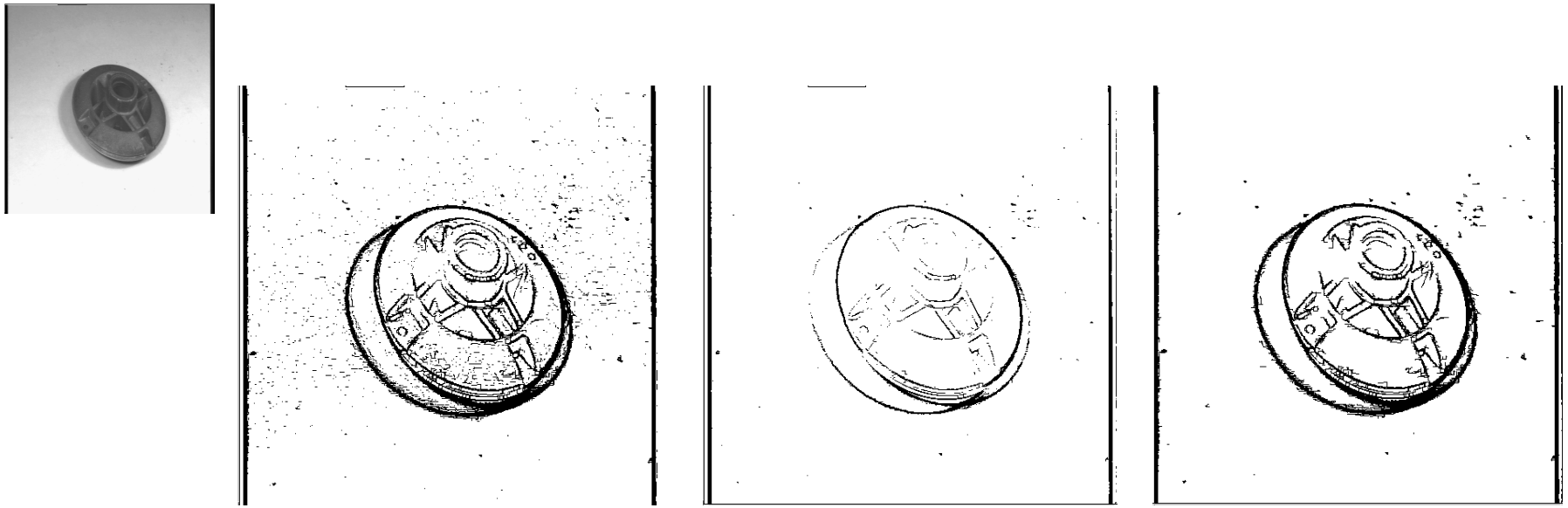


Thresholding with Hysteresis

Hysteresis fills in most of most of the gaps

Problem: pixels along this edge didn't survive the thresholding

Thresholding with Hysteresis



Low threshold High threshold Hysteresis

What Canny Did

- Showed that 1st derivative of a Gaussian smoothed image is the optimal way to detect step edges in noise
  - Explained why 1st derivatives are a good idea
- Designed the industry standard thresholding method
  - Non-maximal suppression
  - Thresholding with hysteresis
- Effectively solved the edge detection problem

COMP2005

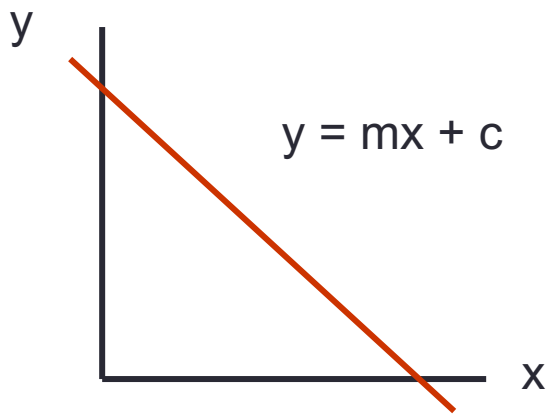
The Hough Transform

Line Parameters

- The standard equation of a line is

$$y = mx + c$$

- If we know the line parameters m,c we can vary x, compute y and draw the line
- This line represents the set of (x,y) pairs that satisfy the above equation



But we don't know the line parameters - we just have some data points (x_i,y_i)

Parameter Space

- But if a data point (x_i,y_i) lies on a line then that line's parameters m,c must satisfy

$$y_i = mx_i + c$$

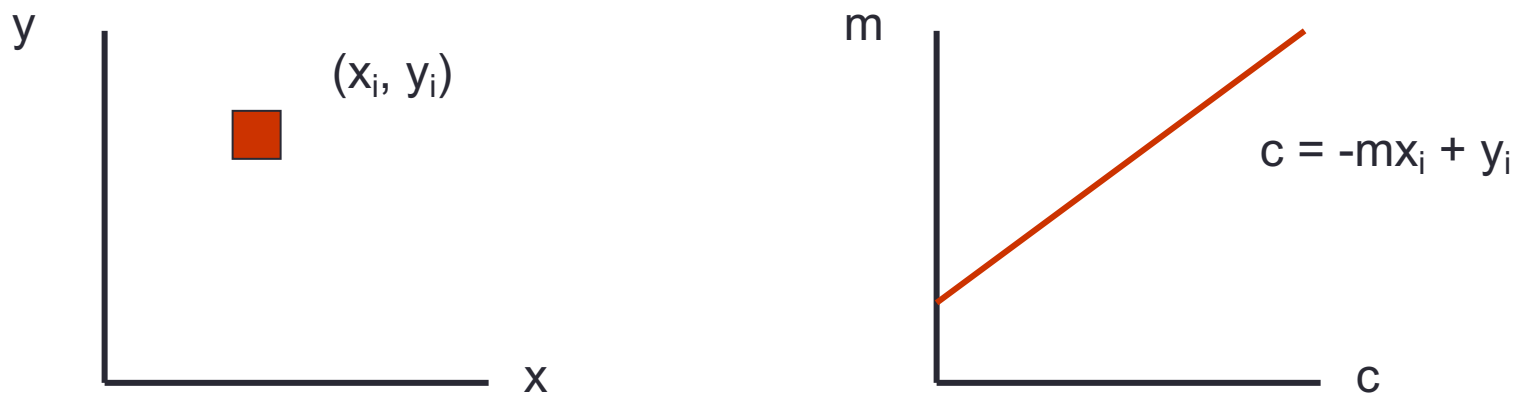
- So we can represent all possible lines through (x_i,y_i) by the set of (m,c) pairs that satisfy this equation

- Rearranging the equation gives

$$c = -mx_i + y_i$$

- This also describes a straight line, but as x_i and y_i are known and m and c are unknown that line is in m,c space – a parameter space

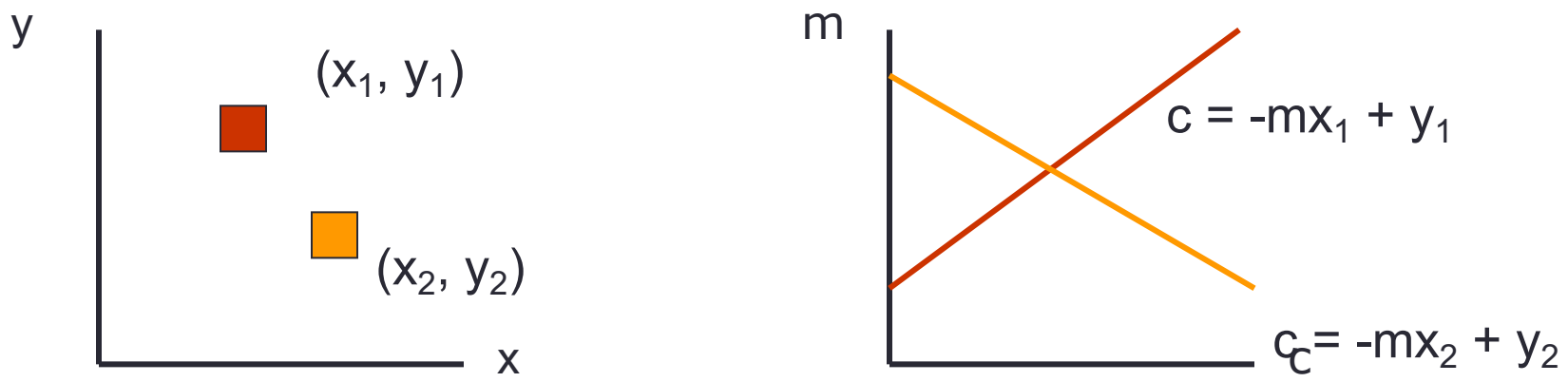
Parameter Space



- This straight line in m,c space represents all the values of (m,c) that satisfy $y_i = mx_i + c$, and so the parameters of all the lines that could pass through (x_i, y_i)

Parameter Space

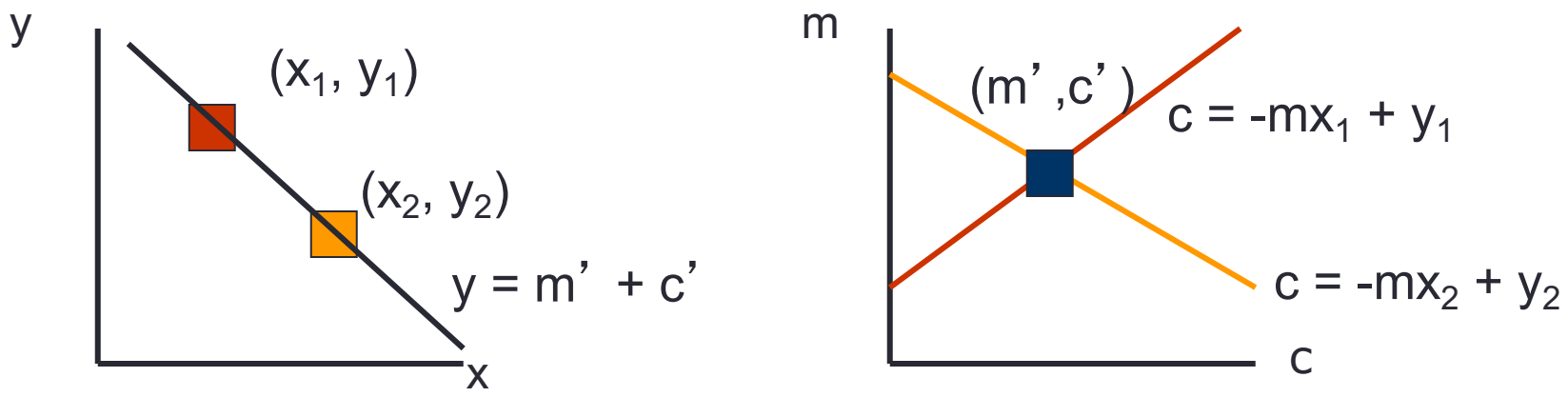
- Suppose there are 2 edges in our image (x_1,y_1) and (x_2,y_2)



- They will each generate a line in (m,c) space representing the set of lines that could pass through them

Parameter Space

- The intersection of our 2 lines is the intersection of 2 sets of line parameters



- The point of intersection therefore gives the parameters of a line through both (x_1,y_1) and (x_2,y_2)

Parameter Space

- Taking this one step further
  - All pixels which lie on a line in (x,y) space are represented by lines which all pass through a single point in (m,c) space
  - The point through which they all pass gives the values of m' and c' in the equation of the line: y=m'x+c'.
- To detect lines all we need to do is transform each edge point into m,c space and look for places where lots of lines intersect

This is what Hough Transforms do

The Hough Transform

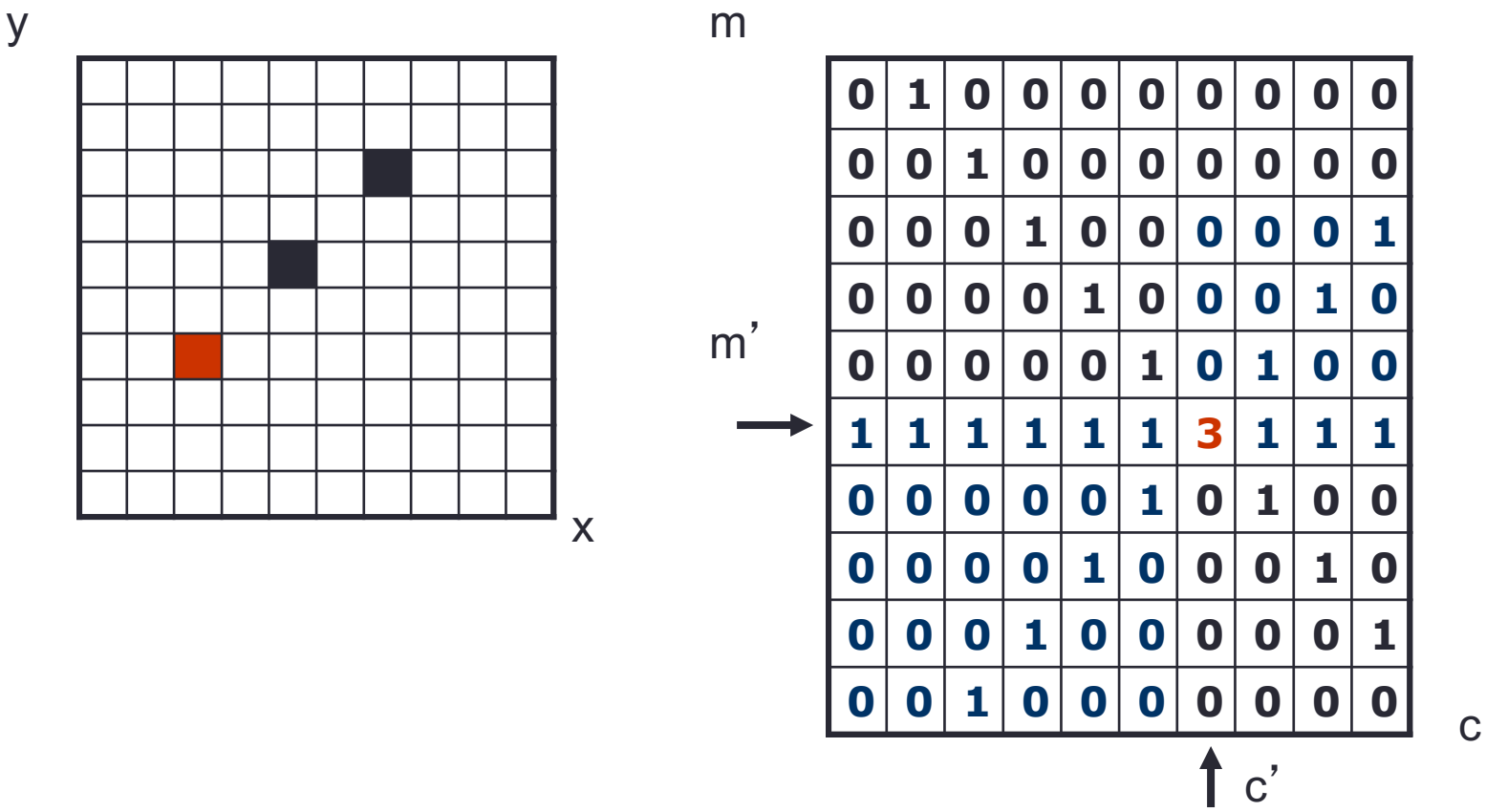
1. Quantise (m,c) space into a two-dimensional array A for appropriate steps of m and c

2. Initialise all elements of A(m,c) to zero

3. For each pixel (x_i,y_i) which lies on some edge in the image, we add 1 to all elements of A(m,c) whose indices m and c satisfy $y_i=mx_i+c$

4. Search for elements of A(m,c) which have large values - each one found corresponds to a line in the original image

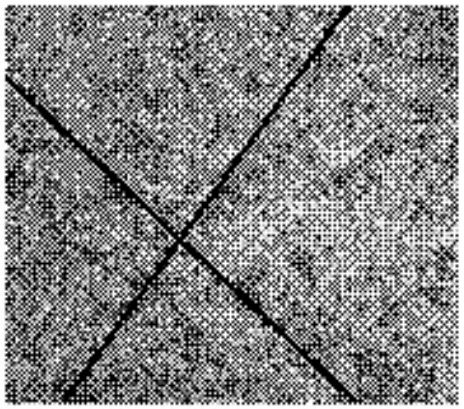
The Hough Transform



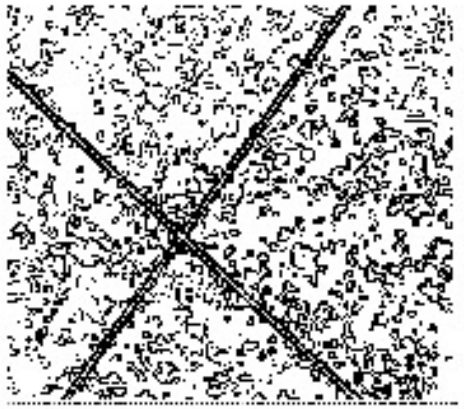
There is one line: y = m' x + c'

A (More) Real Example

Image



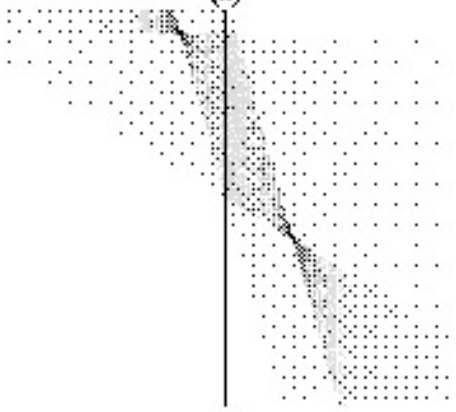
(a)



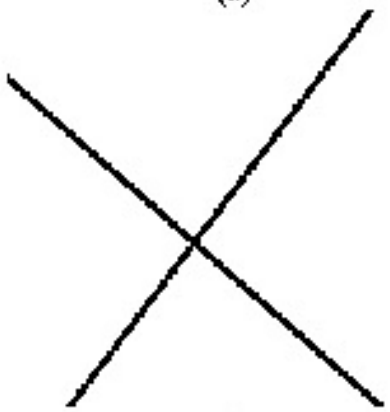
(b)

Edges

Accumulators



(c)

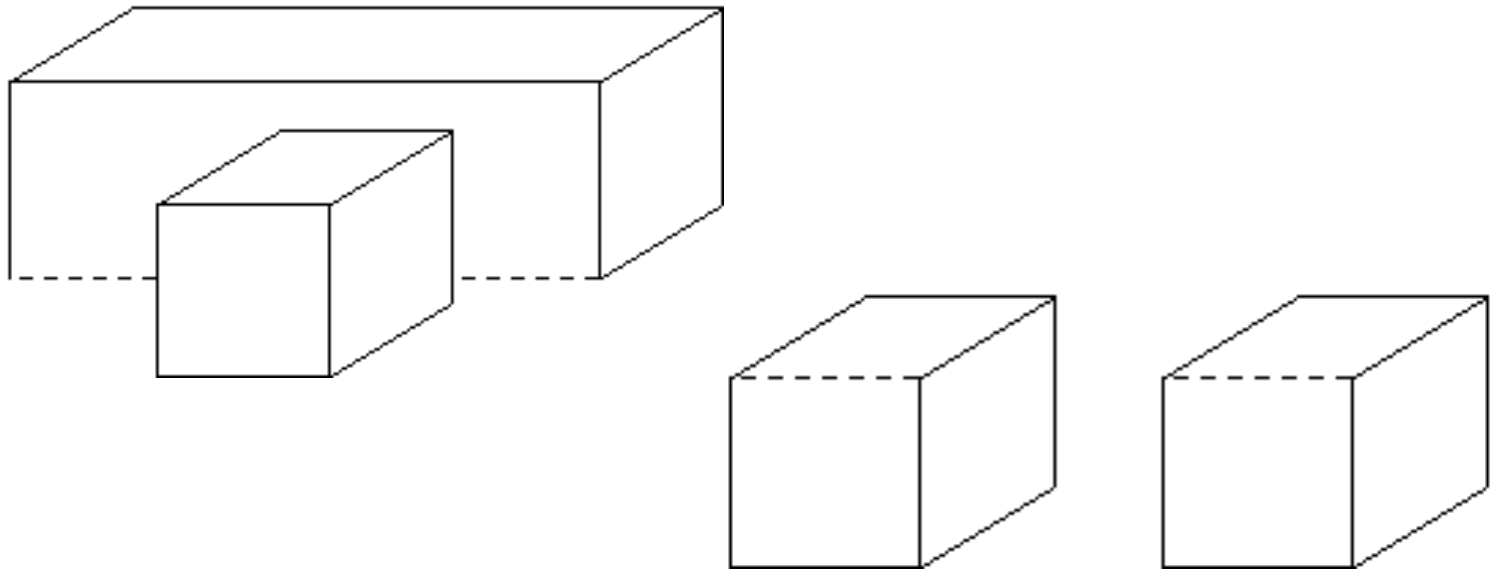


(d)

Result

Line Parameters

- The Hough Transform only supplies the parameters of the lines it detects; this can be an advantage or a disadvantage



COMP2005

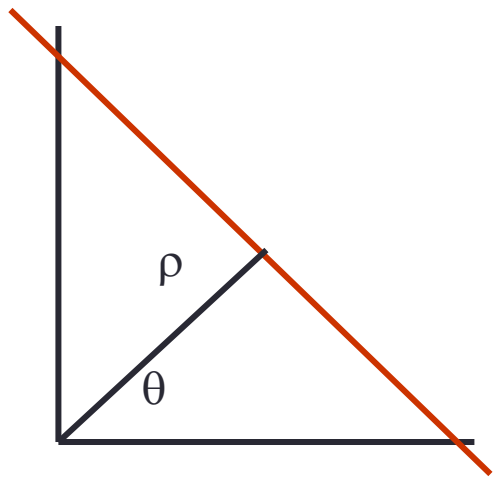
Other Parameter Spaces

Straight Lines Again

- Another equation for a straight line

$$\rho = x.\cos(\theta) + y.\sin(\theta)$$

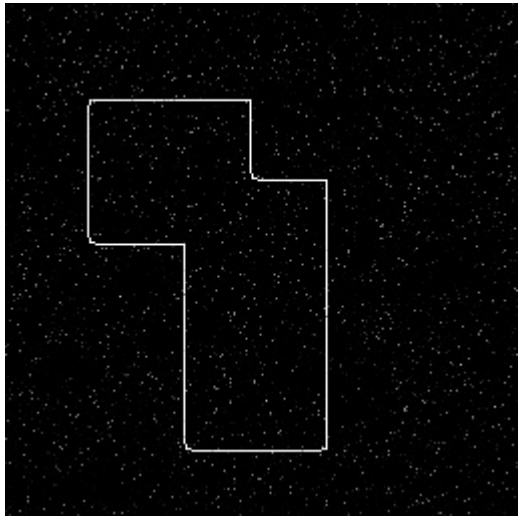
- For an n x n image
  - ρ is in range [0, 2n]
  - θ is in range [0, 2Pi]

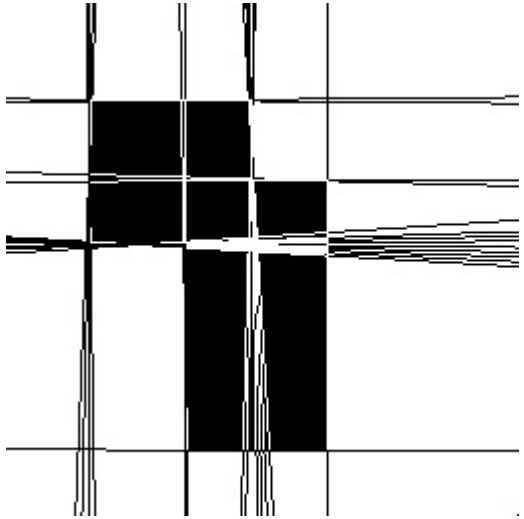


- Each x_i, y_i generates a sinusoid in ρ, θ space

A ρ, θ Example

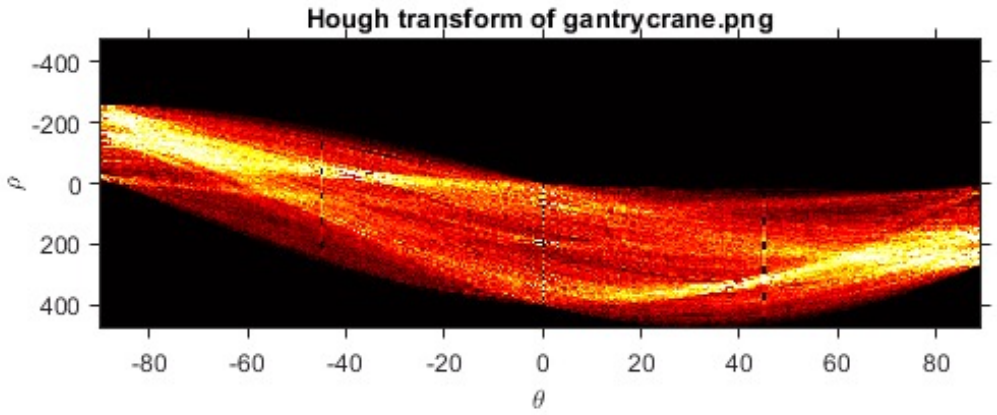
A ρ, θ Example





A key question is what happens to peaks in parameter space when noise or occlusion arise?

The Hough Transform in Matlab

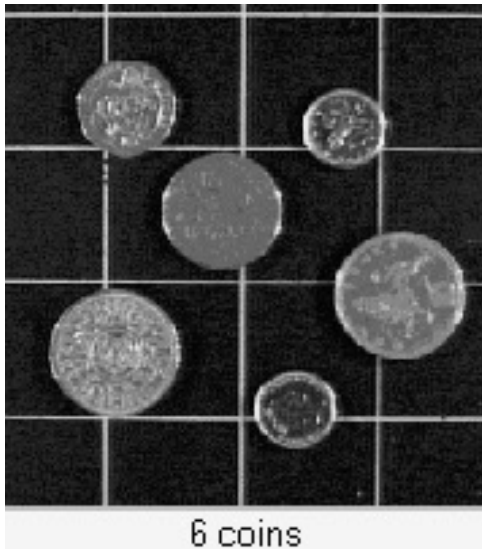


Like many implementations, Matlab's Hough allows you to specify which part of the parameter space you're interested in…..

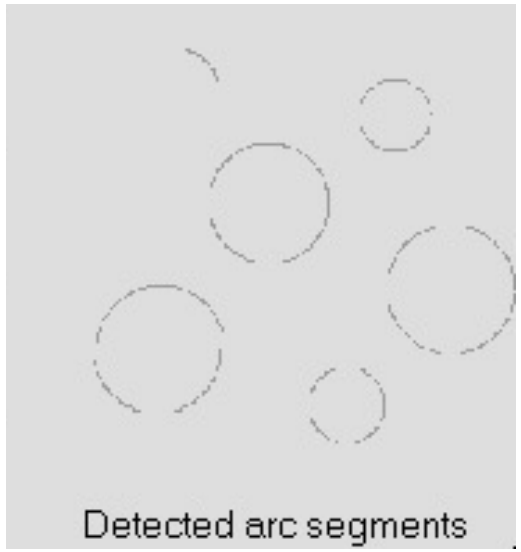
https://uk.mathworks.com/help/images/ref/hough.html

Circles

- Parametric equation is $r^2 = (x-a)^2 + (y-b)^2$
  - - full problem $\rightarrow$ cones in (a, b, r) space
  - - known r $\rightarrow$ circles in (a,b) space



6 coins



Detected arc segments

What about ellipses?

The Hough can in principle be applied to any parameterizable curve, though its not always practical