So far..

Multiscale image representation

- critical points, top points, corners
- SIFT, SURF

Next..

Extracting edges and lines

Edge and line detection

What is an edge?

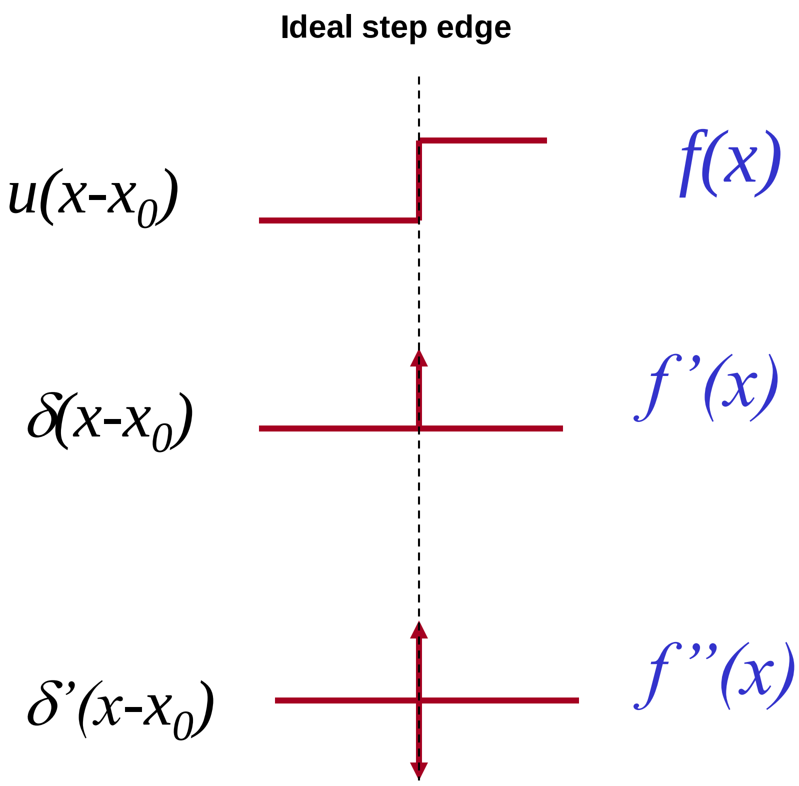
- Edge ≃ change in intensity profile

1-D intensity profiles

Roof type

Step type

Derivatives of an ideal edge



Derivatives of an ideal edge.. contd.

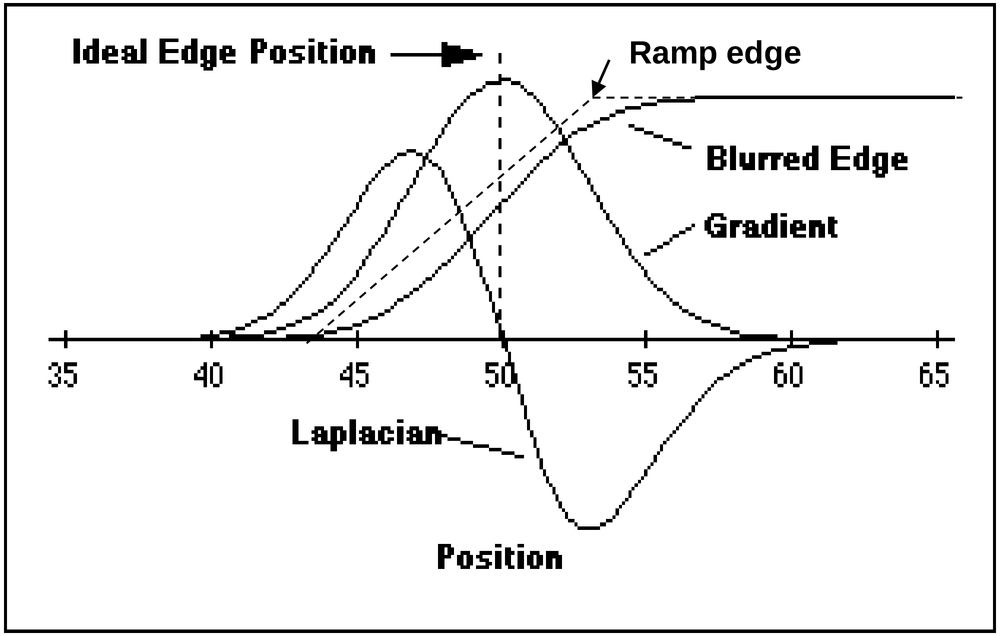
Assume a ramp function for the edge:

$f(x) = r(x-x_0) - r(x-x_1);$

$f'(x) = u(x-x_0) - u(x-x_1);$

$f''(x) = \delta(x-x_0) - \delta(x-x_1);$

In reality, the edge is often a blurred version of the ramp



Characterising an edge

- **Step** type
  - Discontinuity in the intensity profile
  - One peak in the gradient
  - 2 peaks of opposite signs in the 2nd derivative
    - i.e. a zero-crossing

- **Roof** type
  - Continuous intensity profile
  - Discontinuity in the gradient profile
  - One peak in the 2nd derivative

Edge detection

Edge $\equiv$ Discontinuity (change) in intensity

⇩

Strategies for edge detection

- **Gradient** based
  - ➢ Compute first derivative (gradient) and threshold this image

- **Zero-crossing** based
  - ➢ Compute second derivative, locate zero crossings and threshold this image

Gradient computation

- Given *f(x)* its derivative is $$\frac{df(x)}{dx} = \lim_{\Delta->0} \frac{f(x+\Delta) - f(x)}{\Delta}$$
- Finite difference approximation: $I[m+1] - I[m]$
- Any derivative (difference) operation amplifies noise!
- Smoothing may be prudent before computing gradient

- Gradient is a vector: $|\vec{g}|\angle\vec{g}$
  - Angle indicates direction of maximal change
- An image is a 2-D function $I[m,n]$
- Gradients are generally computed in two orthogonal directions using *n*x*n* masks and results are combined using some norm
- Mask coefficients should sum to ?

Gradient operators

Roberts

0	1
-1	0

-1	0
0	1

Gradient in 2 diagonal directions

Prewitt

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

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

Gradient in 2 orthogonal directions

Sobel

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

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

Smoothing + gradient in orthogonal directions

From gradient to edges

- Gradient is a vector: $\vec{g}[m,n] = \{g_x[m,n], g_y[m,n]\}$

Gradient angle /edge direction

$$\angle g[m,n] = \tan^{-1}\left(\frac{g_y[m,n]}{g_x[m,n]}\right)$$

$$\therefore |\vec{g}[m,n]| = \sqrt{g_x^{\,2}[m,n] + g_y^{\,2}[m,n]}$$

$$OR$$

$$|\vec{g}[m,n]| = |g_x[m,n]| + |g_y[m,n]|$$

Gradient magnitude

- Gradient magnitude $|\vec{g}|$ is a greyscale image or a soft map
- A binary "edge map" can be obtained by thresholding $|\vec{g}|$
  - Large gradient magnitude ➔ strong edge

Second derivative operators

Masks

Laplacian

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

0	-1	0
-1	4	-1
0	-1	0

Laplacian of a Gaussian (LoG)

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

1	-2	1
-2	4	-2
1	-2	1

Mexican hat

Zero-crossing based edge detection

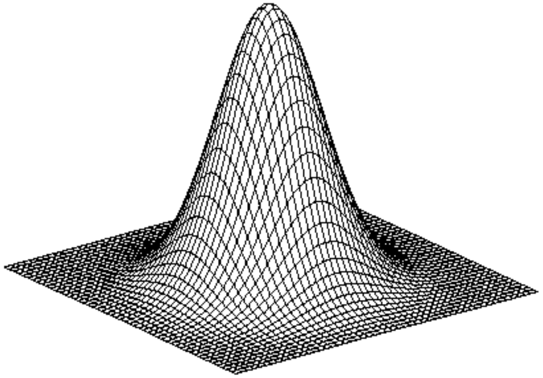
$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- **Laplacian**
  - 2nd derivative operator
  - Isotropic
  - Can only *locate* edges, can't give edge direction
  - Noisy
  - No differentiation between strong vs weak edge

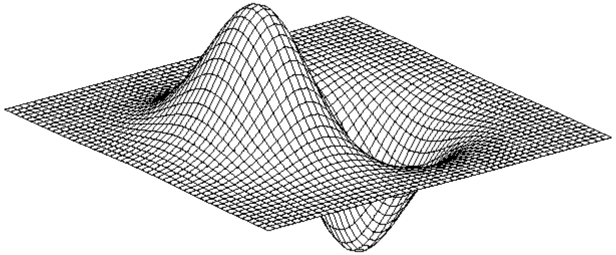
LoG filter kernel

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

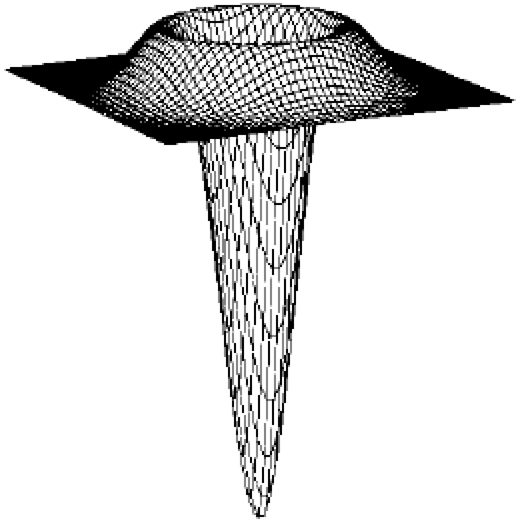
$$LoG(x, y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2 + y^2}{2\sigma^2}\right] e^{-\frac{x^2 + y^2}{2\sigma^2}}$$



G
Smoothing kernel



First derivative of G



Laplacian of G

Laplacian of Gaussian (LoG)

LoG filter

3x3

1	-2	1
-2	4	-2
1	-2	1

LoG with $\sigma = 1.4$

9x9

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Mask size increases exponentially with σ !

Implementation of LoG - as DoG

- LoG can be approximated with Difference of Gaussians (DoG)
  - as in Human visual system (simple cells in V1)

$$\nabla^2 (G * f) = G_{\sigma_1} - G_{\sigma_2}$$

$$\boxed{-\frac{1}{\pi\sigma^4}\left[1-\frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}} = \boxed{\frac{1}{\sqrt{2\pi}\sigma_1}e^{-\frac{x^2+y^2}{2\sigma_1^2}}} - \boxed{\frac{1}{\sqrt{2\pi}\sigma_2}e^{-\frac{x^2+y^2}{2\sigma_2^2}}}$$

Using the Taylor series expansion, the equivalence relation requires $\sigma_1 : \sigma_2$ to be ……

Implementation of LoG - separable kernels

$$\nabla^2 G(x,y) = G_1(x)G_2(y) + G_1(y)G_2(x)$$

$$G_1(x) = \frac{1}{K}\left[1 - \frac{x^2}{2\sigma^2}\right]e^{-\frac{x^2}{2\sigma^2}}$$

$$G_2(y) = \frac{1}{K}e^{-\frac{y^2}{2\sigma^2}}$$

Optimal edge detector

Desirables

a. good detection - need to smooth out the noise
   - More smoothing leads to better noise suppression

b. good localisation – need to avoid or do less smoothing
   - Conflicts with (a)!

c. unique response to an edge – what if there are two maxima?
   - Need to limit the allowable separation between maxima

Canny edge detector

- Derives the filter via optimisation
- Filter is roughly a derivative of a Gaussian

Main steps in Canny edge detector

- ➢ Multiscale Gaussian smoothing of the given image
- ➢ Gradient computation
- ➢ Non-maxima supression
- ➢ Hysteresis thresholding
- ➢ Rejection of weak edges not connected to strong edges

Canny, *PAMI 1986*

Canny edge detector.. contd.

Non-maxima suppression applied to $|g|$

1. Quantise the edge direction to 4 directions
2. Compare $|g(x,y)|$ with its neighbours in edge normal directions

    If $|g|$ is less than either, suppress, else keep the pixel as edge

Hysteresis thresholding applied to $|g|$

1. If $|g| \geq t_{high}$ then strong edge
2. If $t_{high} \geq |g| \geq t_{low}$ then weak edge ; t_{high} is usually 2 to 3 times t_{low}

Region labelling

– Reject regions without strong edge pixels

Performance

\+ Very good results <u>with proper tuning</u>

\- Computationally more expensive than Sobel, Prewitt etc.

Edge based segmentation

Input image

Gradient image

Issues in edge based segmentation

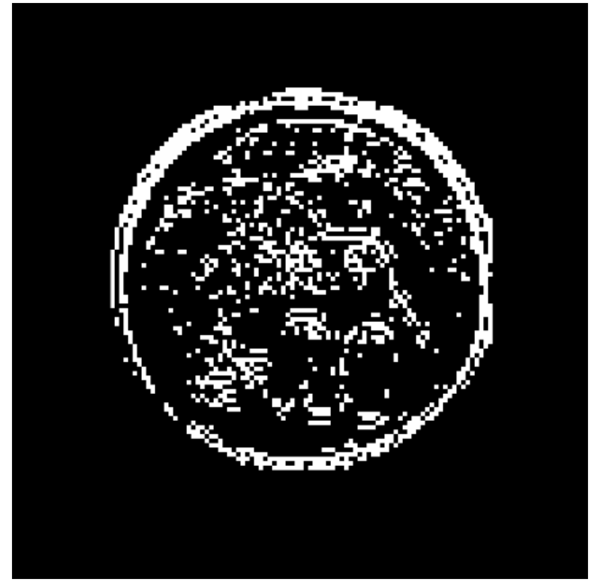
Noisy input image

Gradient image

Sensitivity to noise

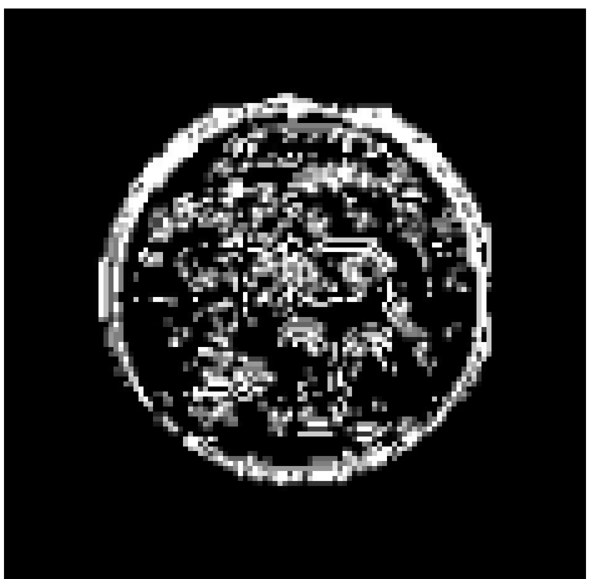
Comparison of edge detection methods

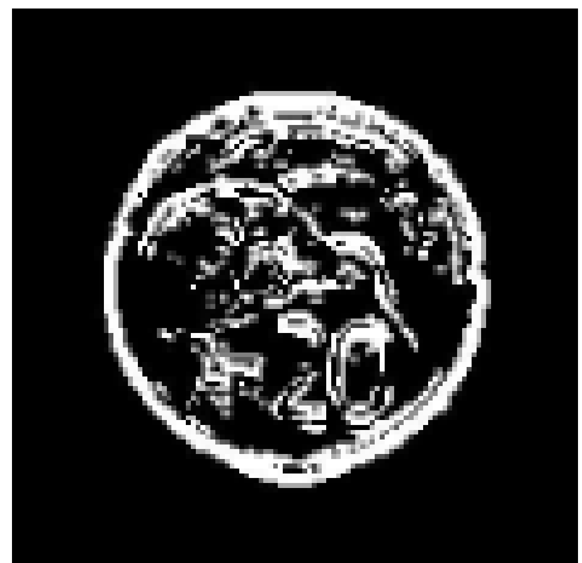
Coin image



Prewitt

LoG





Canny

Effect of σ in Canny edge detection

original

Canny with $\sigma = 1$

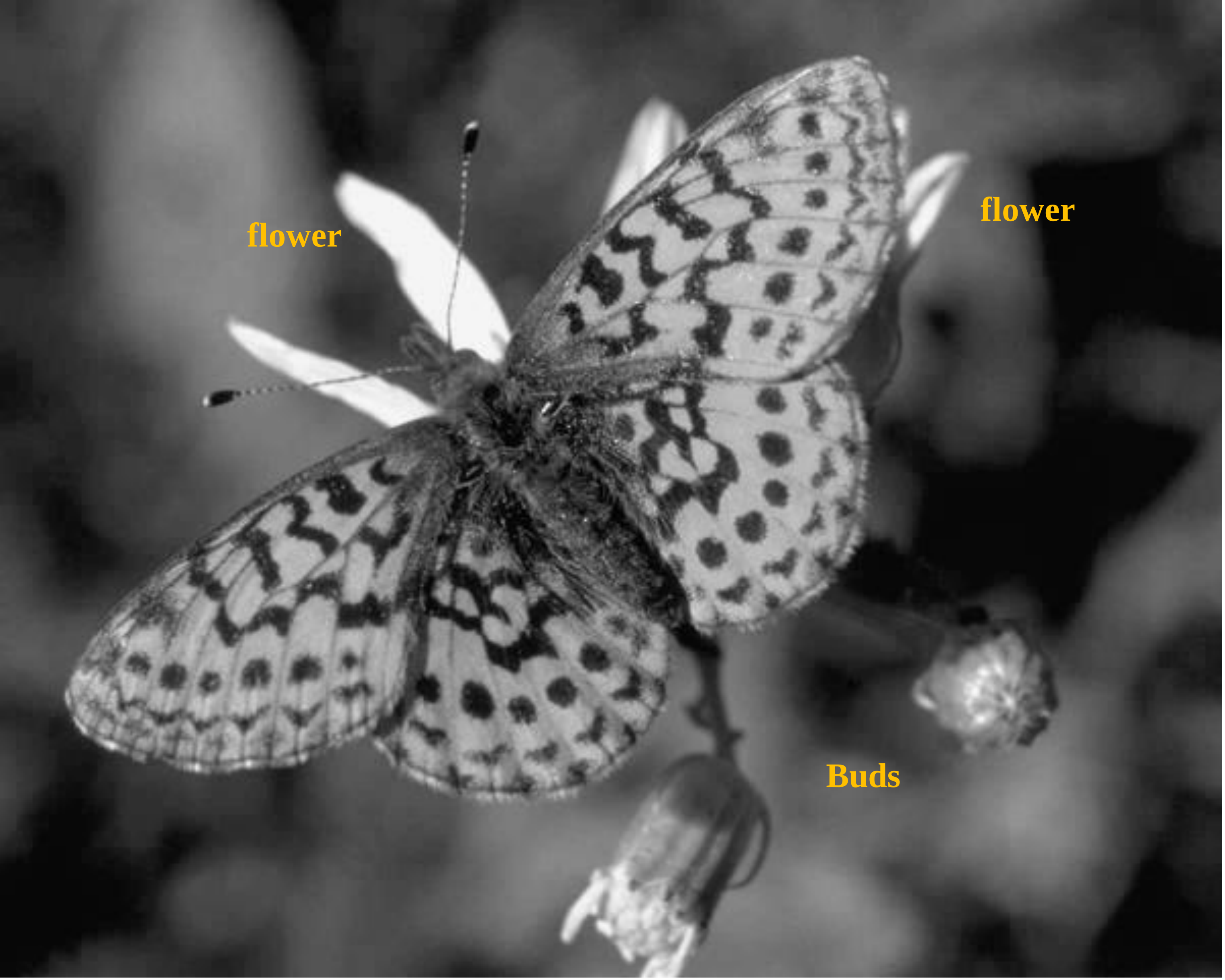
Canny with $\sigma = 2$

The choice of Gaussian kernel size σ controls the output

- small σ helps detect fine features
- large σ helps detect large scale edges

Canny performs edge detection in <u>scale-space</u>
about which we will learn about later

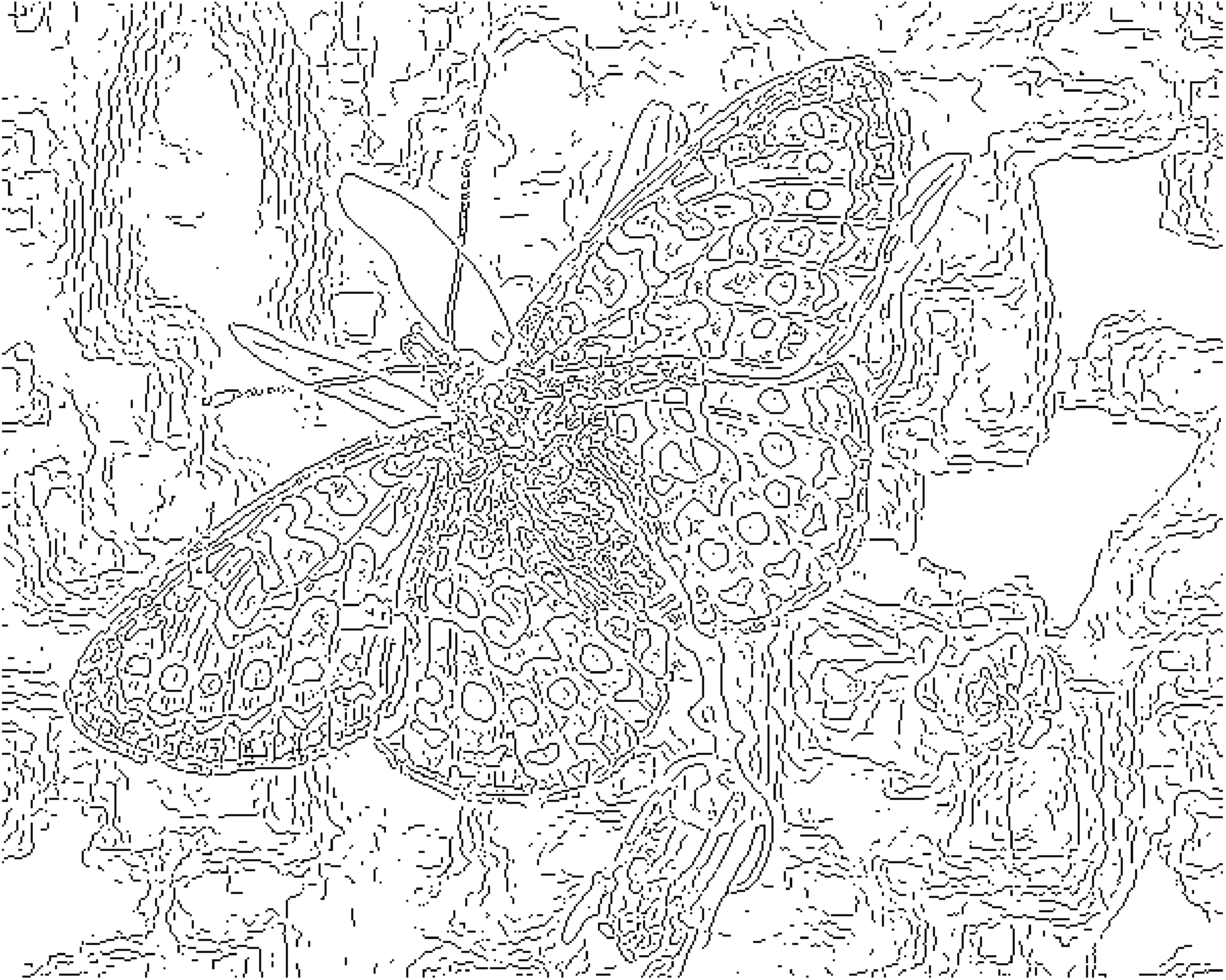
An application



Task: Recognise the butterfly

Idea: do edge detection → extract the shape → match to a model

Potential problems – flower and buds; markings on the butterfly



Canny output: fine scale and high threshold
- too noisy an edge map

coarse
scale,
high
threshold

Canny output: coarse scale and high threshold
- too sparse an edge map to extract shape

Finding oriented edges

What if we need to identify edges in a specific direction ?

Compass operators

- Detect edges in specific directions
- Angular resolution $\propto$ mask size

Examples

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

North

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

North-west

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

South-west

Post processing – edge linking

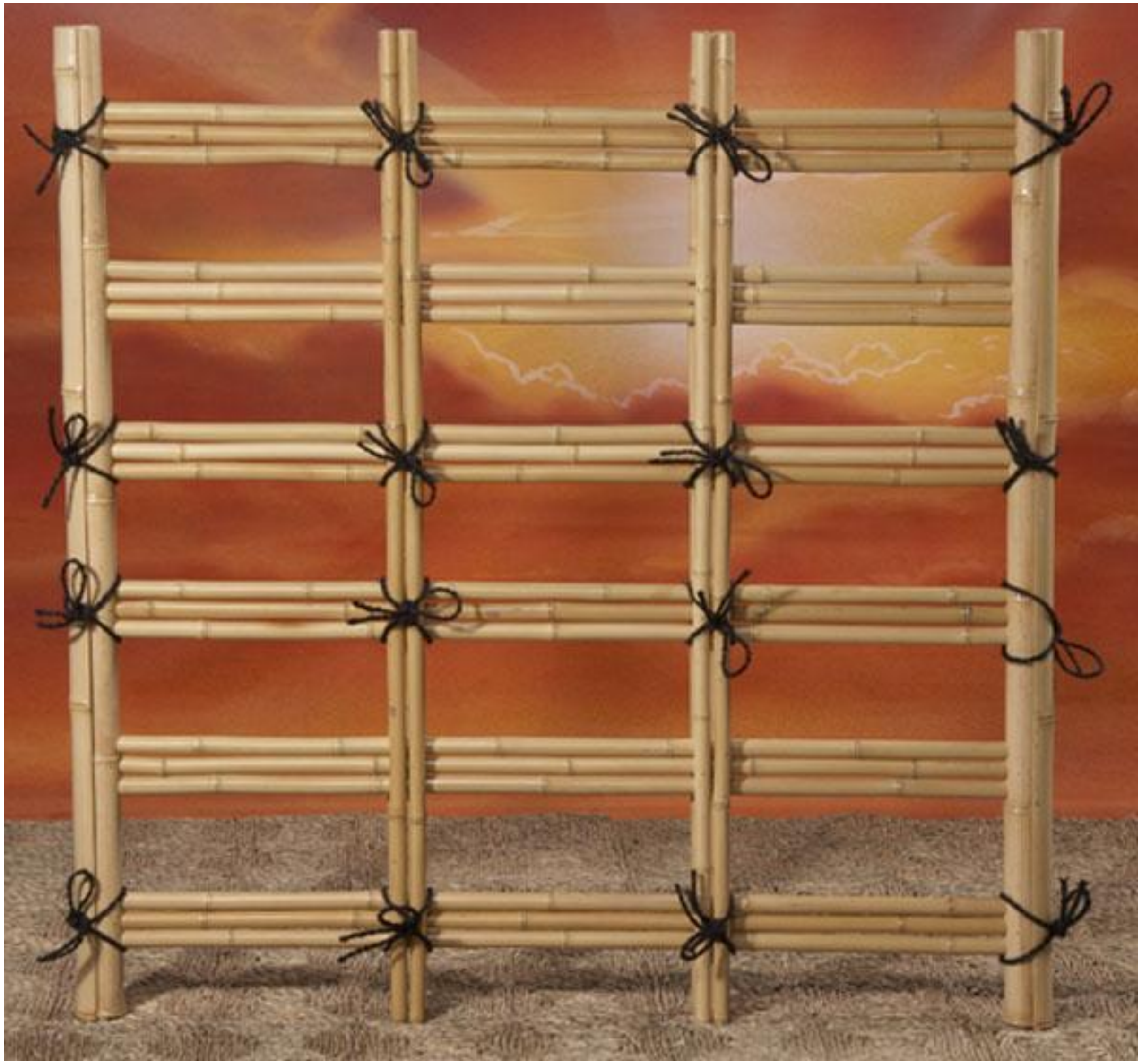
To obtain linked edges

Algorithm

1. Start from any edge pixel.
2. Examine its neighbourhood with a 3x3 window
3. Add any edge pixel which has similar strength and direction
4. Shift the centre of the window to the added pixel position
5. Repeat steps 3-4 for every edge pixel in the image

Finding lines and circles

Count the number of poles



Soln: Find lines and count them?

How do you find lines in an image?

Approach 1: Matched filter

- Design a mask for line in the desired orientation
- Convolve the mask with the image and find peaks

Demerits:

- Orientation of lines present, is generally unknown
- Multiple masks required to detect all lines
- Sensitivity to noise, missing pixels/occlusion

Back to basics - representing a line

Let point $P = (x,y) \in \mathrm{R}^2$

Consider a line through $P \in \mathrm{R}^2$

How do we represent a line?

1. $y(x) = mx + c$ ➔ slope intercept form
2. $Ax + By + 1 = 0$ ➔ homogeneous form
3. $r(\phi) = x \cos \phi + y \sin \phi$ ➔ normal form

$line \equiv P_l : (x_l, y_l)$ a set of collinear points $c_0 = x_l\, m_0 + y_l$

$$line \equiv (x_l, y_l) \text{ with } (m_0, c_0) \equiv (r_0, \varphi_0)$$

Different ways to think of a line

A set of collinear points $\in$ (x,y) space

is equivalent to

A set of lines intersecting at (m_0,c_0) space

or to

$\boxed{\text{A set of sinusoids intersecting at one point in } (r,\varphi) \text{ space}}$

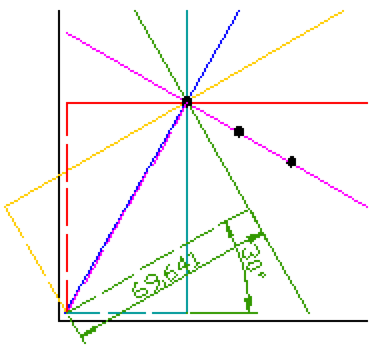
Hough transform: detect collinear points in (x,y) space
by detecting concurrent curves in (r,φ) space

Hough Transform – geometrical view

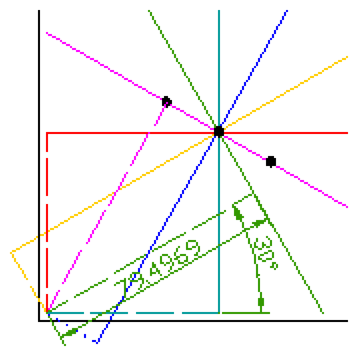
Constraints:

(x,y) → [m,n]; positive valued

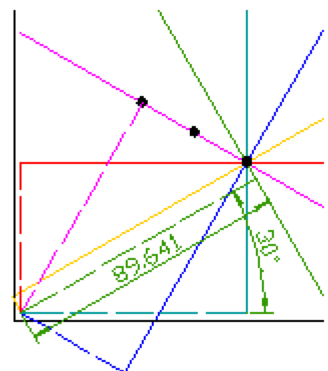
(r,φ) → [i,j]; positive valued



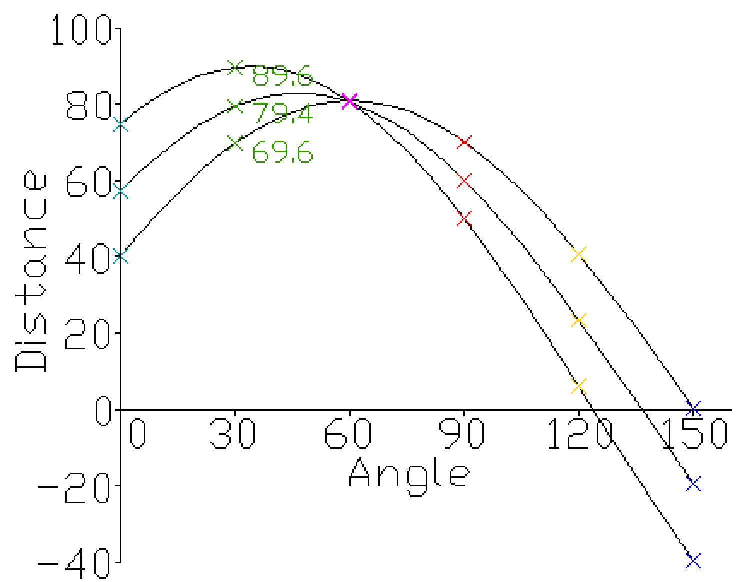
Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4



Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	−19.5



Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	−39.6



From wikipedia

Hough transform- algorithmic view

Strategy for detection: *evidence gathering*

- Map given image points to Hough parameter space
  - Accumulator array
- Most common use is to detect lines, circles and ellipses

Merits

- Less sensitive to noise
- Can fill in automatically

Demerit

Can be computationally expensive for non-linear shapes

How does a point in image space vote?

$$w = x\cos(\phi) + y\sin(\phi)$$

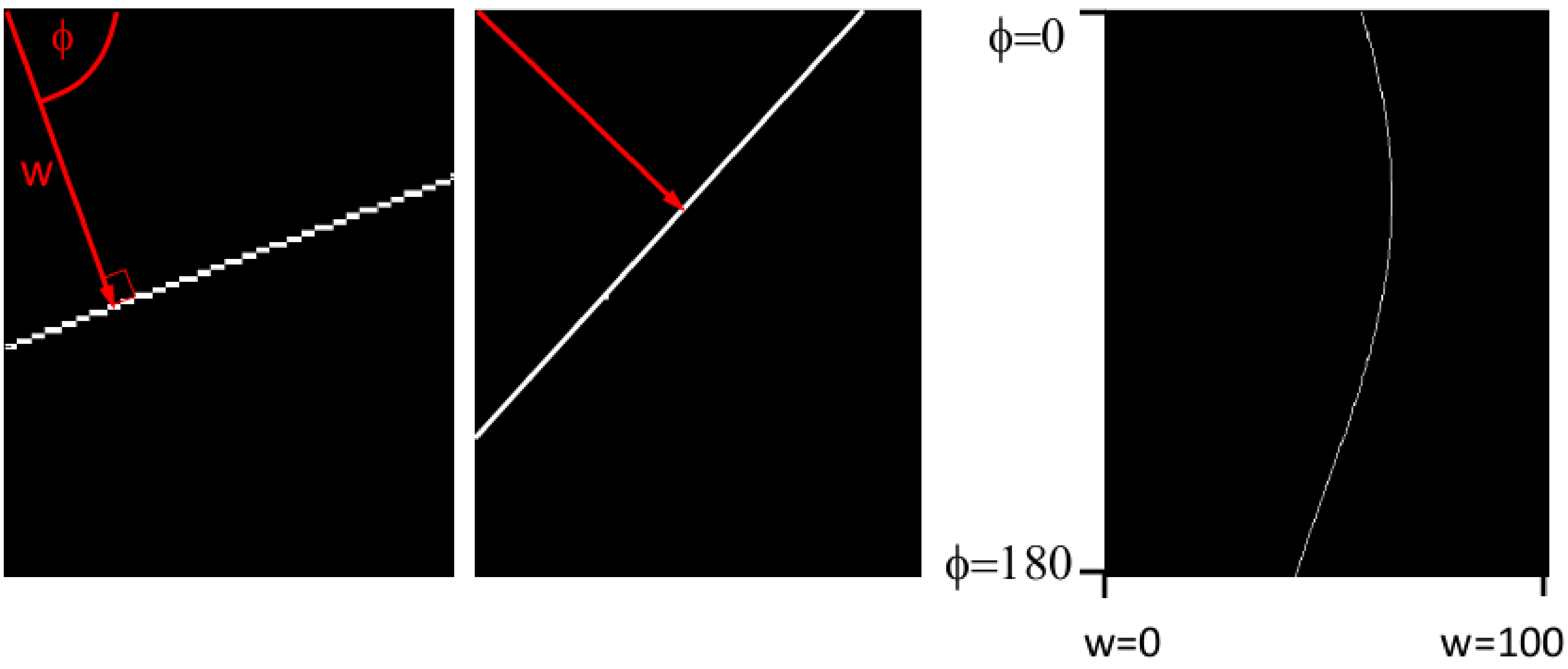
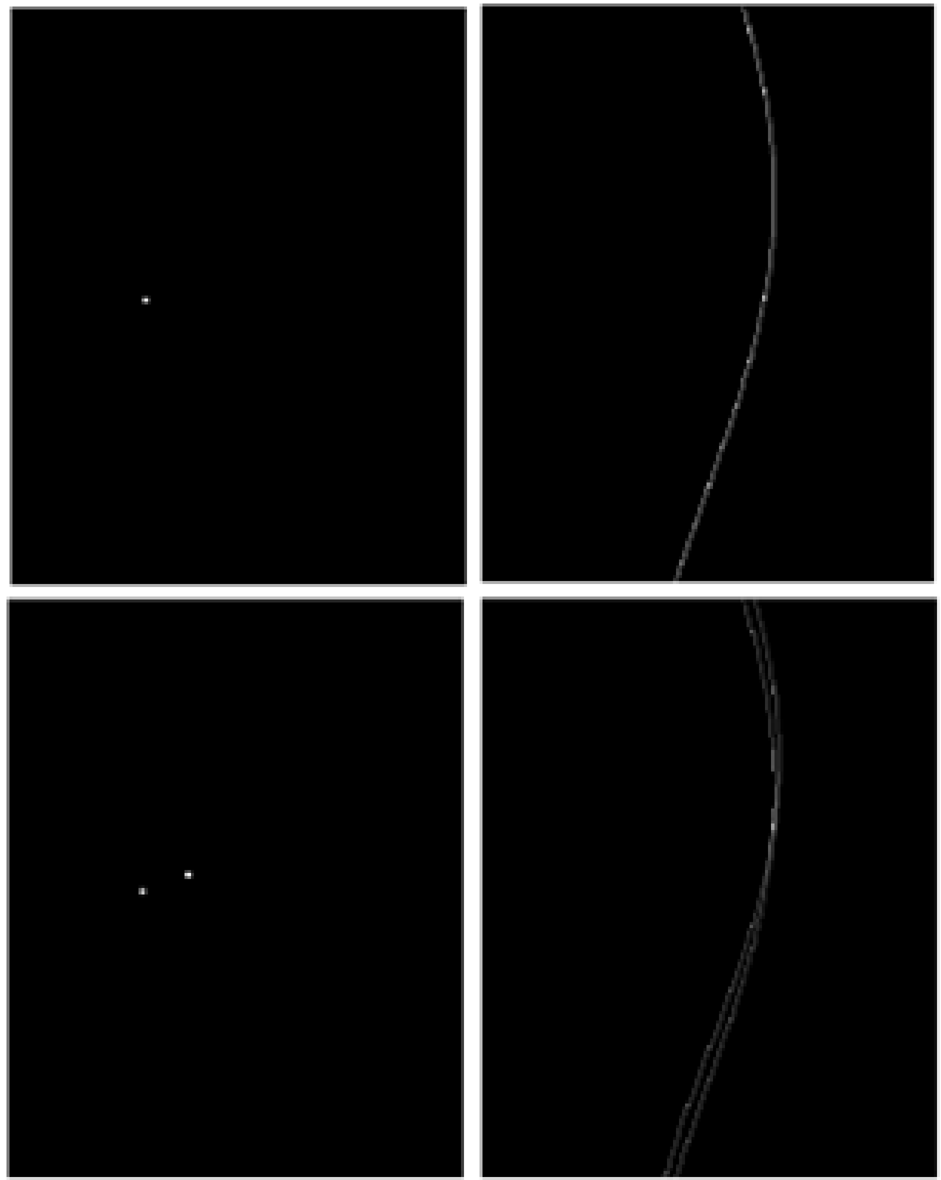
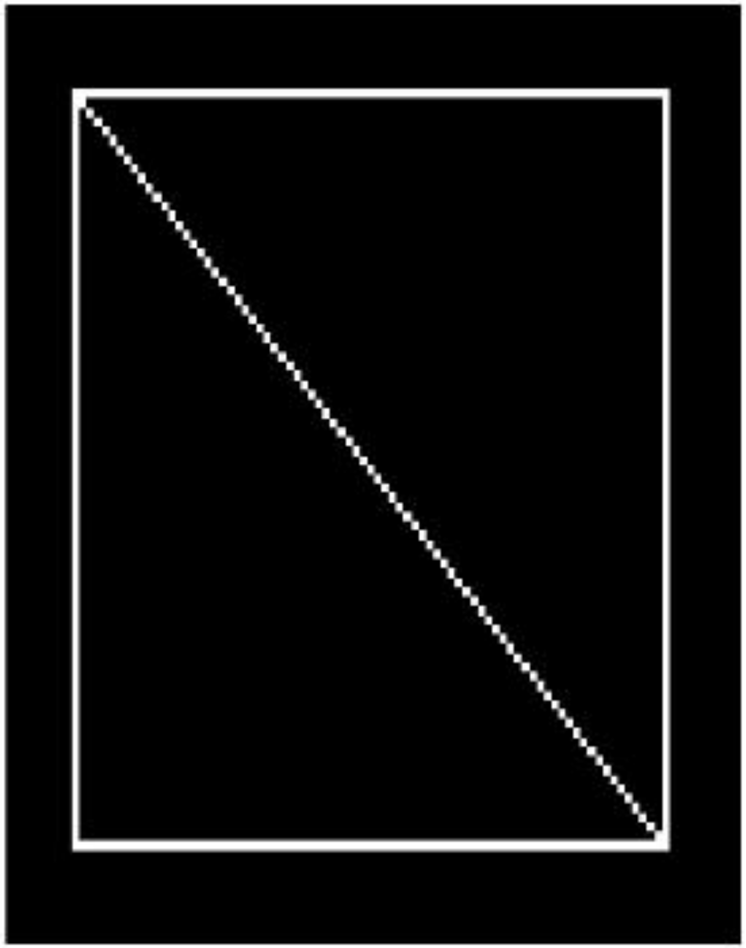
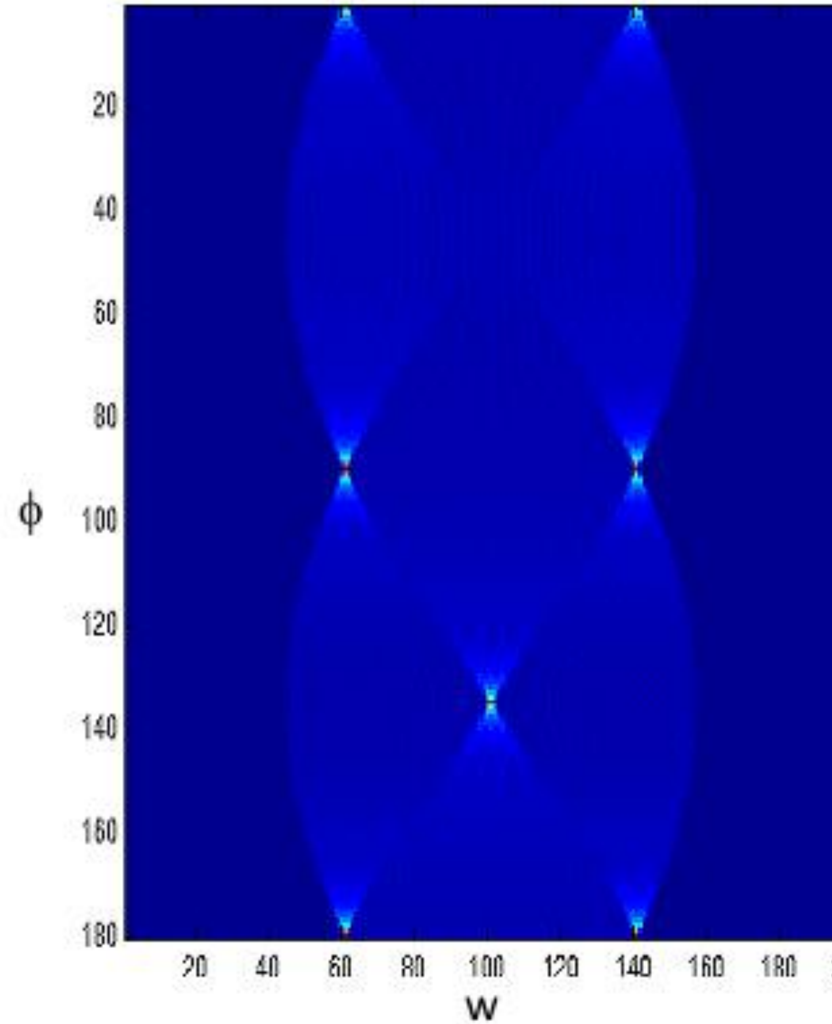


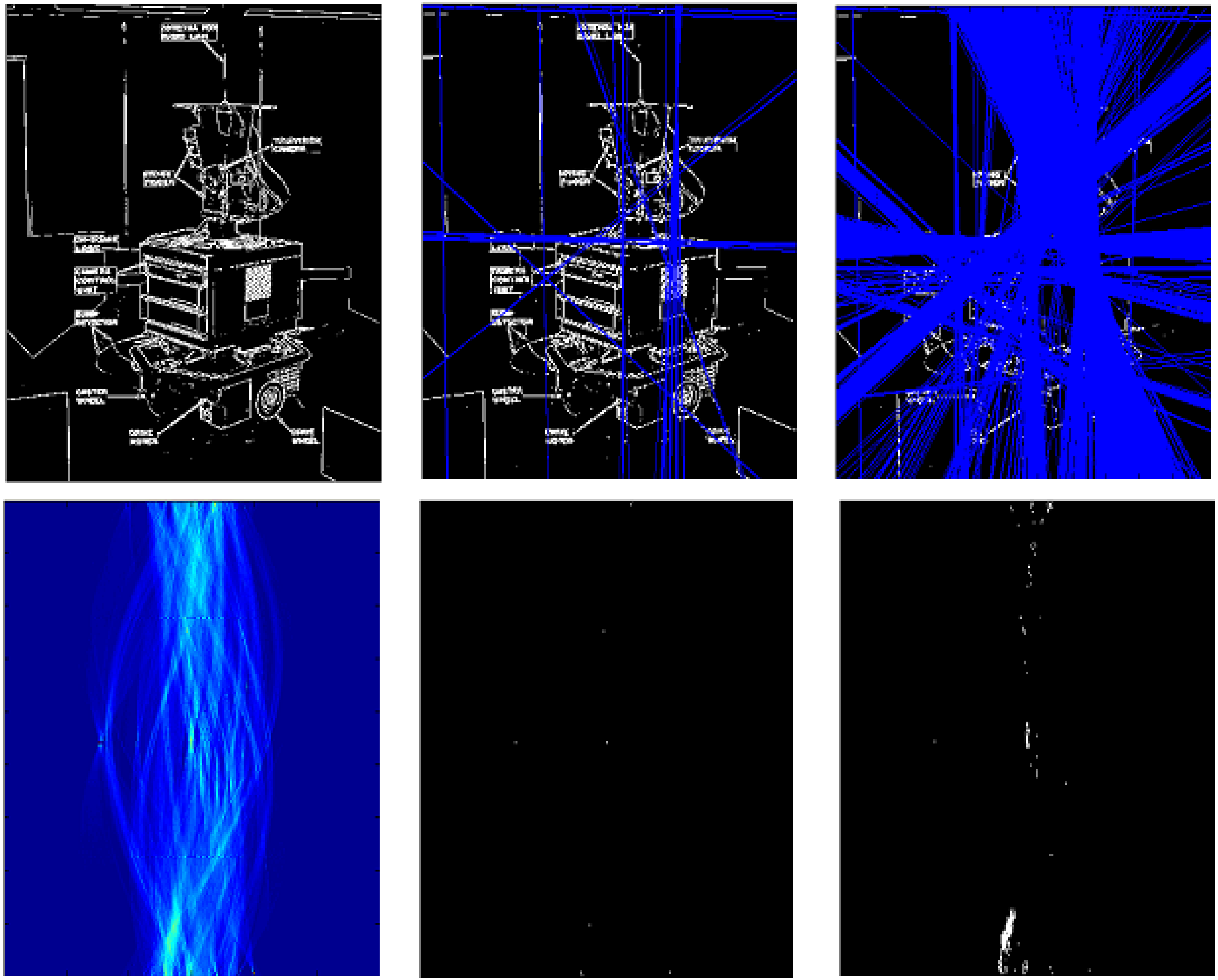
Image space Hough space



A simple example







Hough transform algorithm

To detect lines:

1. Initialize H[r, θ] = 0

2. For each edge point in I[x,y]

   for θ = 0 to 180

   $r = x\cos\theta + y\sin\theta$

   Increment H[r, θ] by 1

3. Find *max* H[r,θ]. Let it be at (r_0,θ_0)

4. The detected line in the image is given by

$$r_0 = x\cos\theta_0 + y\sin\theta_0$$

Issues

- number of bins in Accumulator array
- thresholding the Accumulator array
- sensitivity to noise

Extensions

Extension 1: Use the image gradient

Extension 2

- give more votes for stronger edges

Extension 3

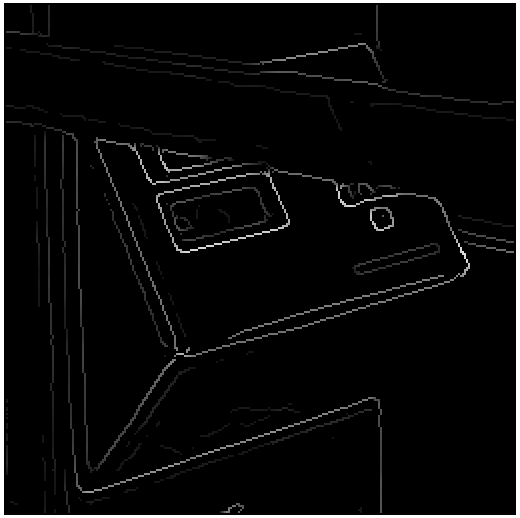
- change the sampling of (r, θ) to give more/less resolution

Extension 4

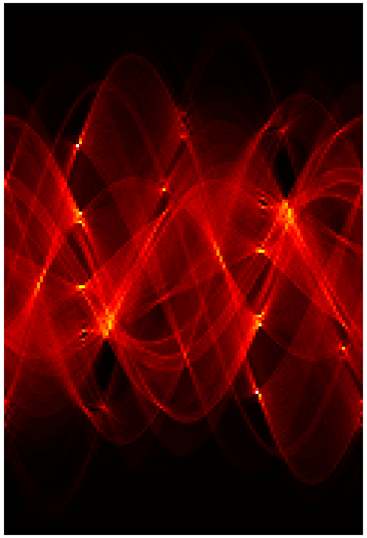
- The same procedure can be used with circles, squares, or any other shape

Example

Original

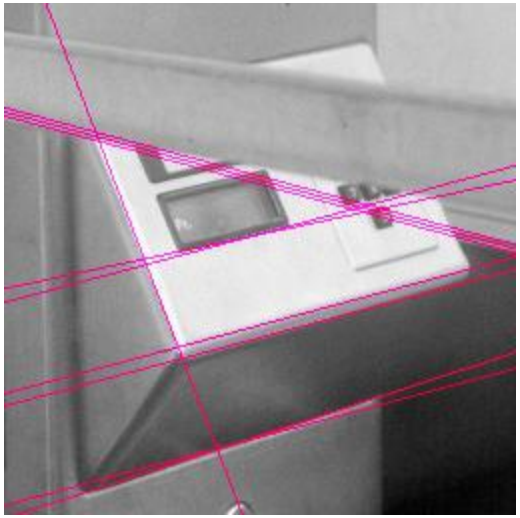


Edge Detection



Parameter Space

Detected Lines

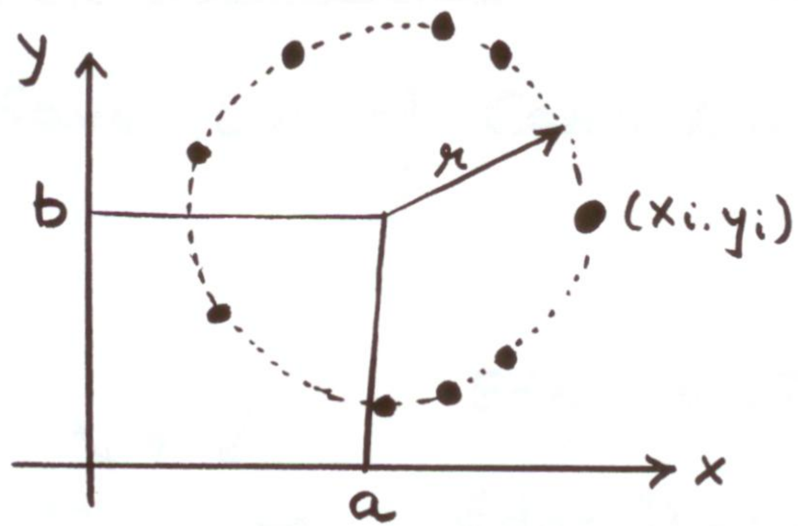


Extension to other Shapes

Finding Circles by Hough Transform

Equation of Circle:

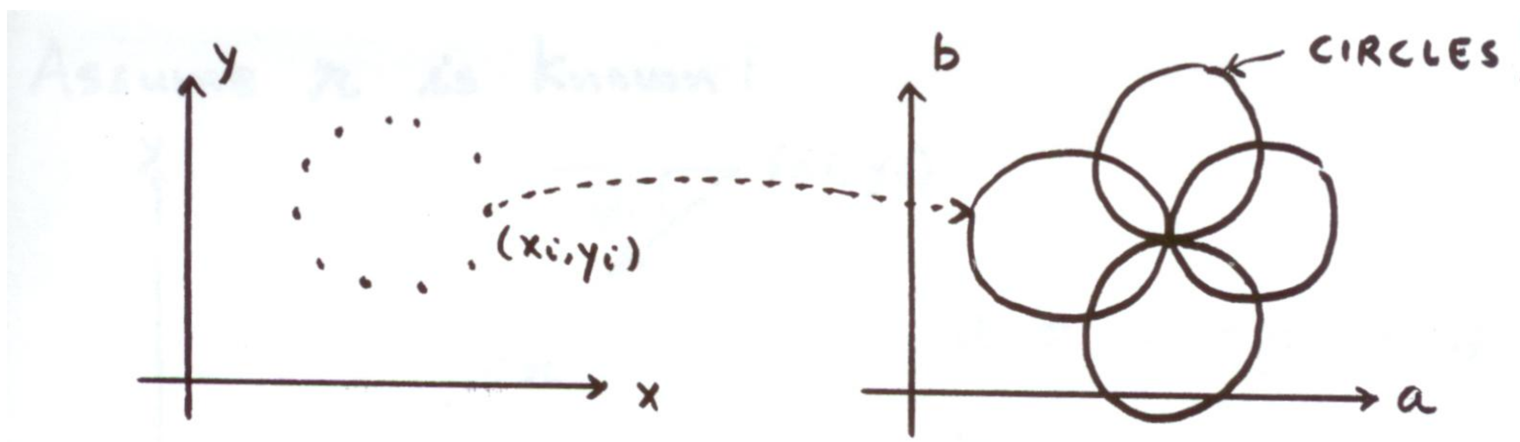
$$(x_i - a)^2 + (y_i - b)^2 = r^2$$



If radius is known — 2D Hough Space

Else — 3D Hough Space

Accumulator Array $A(a,b)$



Each point (x_i,y_i) becomes a circle in a-b space centred at (x_i,y_i)

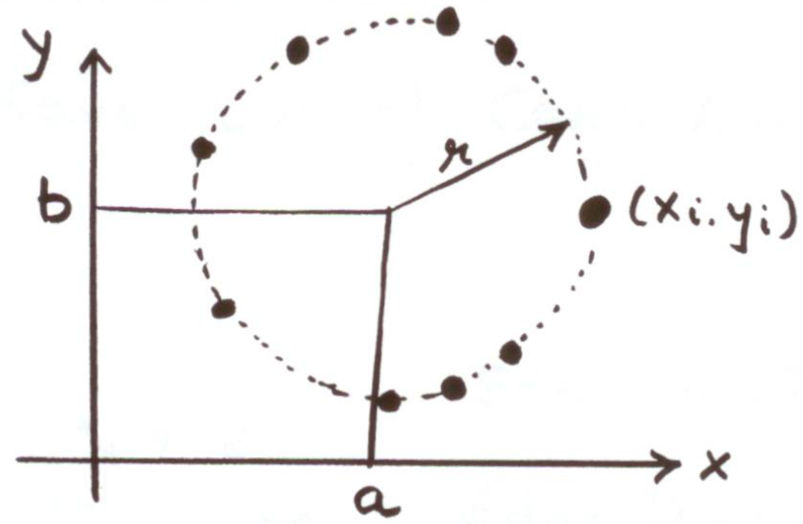
Finding Circles by Hough Transform

Equation of Circle:

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

$$x_i = a + r\cos\theta \Rightarrow a = x_i - r\cos\theta$$

$$y_i = b + r\sin\theta \Rightarrow b = y_i - r\sin\theta$$



Every point (x_i,y_i) becomes a circle in (a,b) space centred at (x_i,y_i)

In *A(a,b,r) e*very point on this circle, forms the apex of a cone with height of the cone being *r*

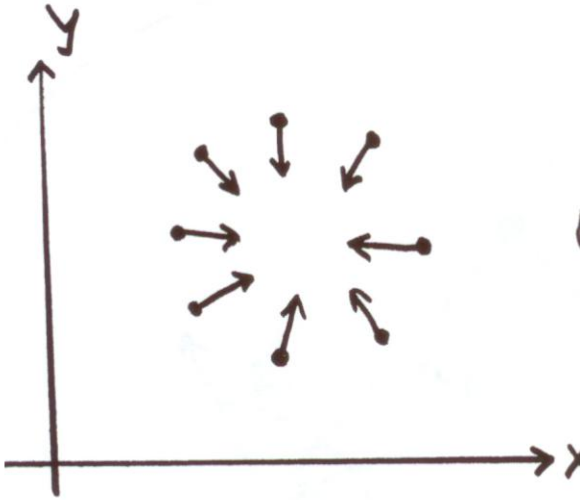
Note: θ is not a free parameter. It defines the trace of the curve (psf)

Using Gradient Information

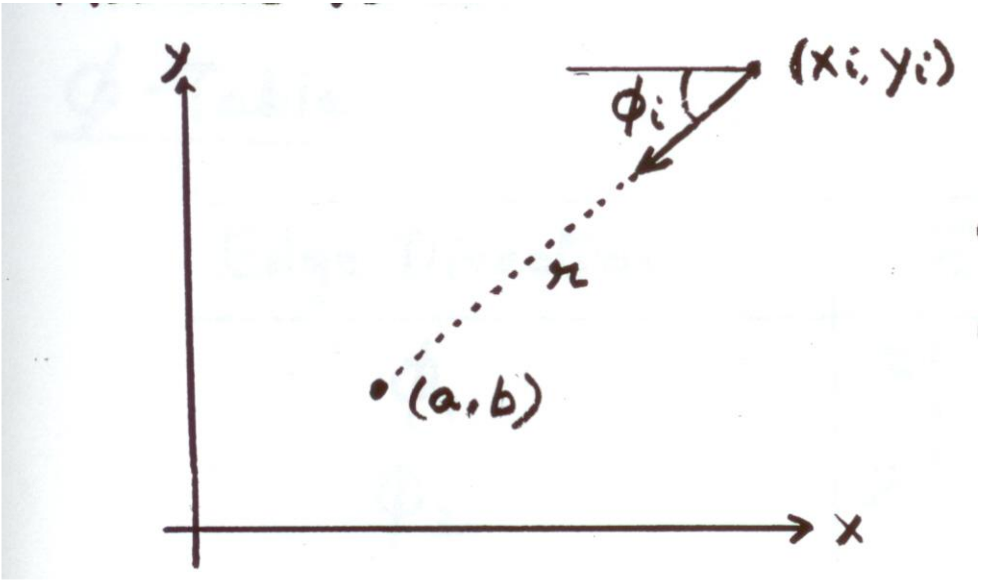
Gradient information can save lot of computation:

Edge Location (x_i, y_i)

Edge Direction ϕ_i



Assume radius is known:



$$a = x - r\cos\phi$$

$$b = y - r\sin\phi$$

Need to increment only one point in Accumulator!!

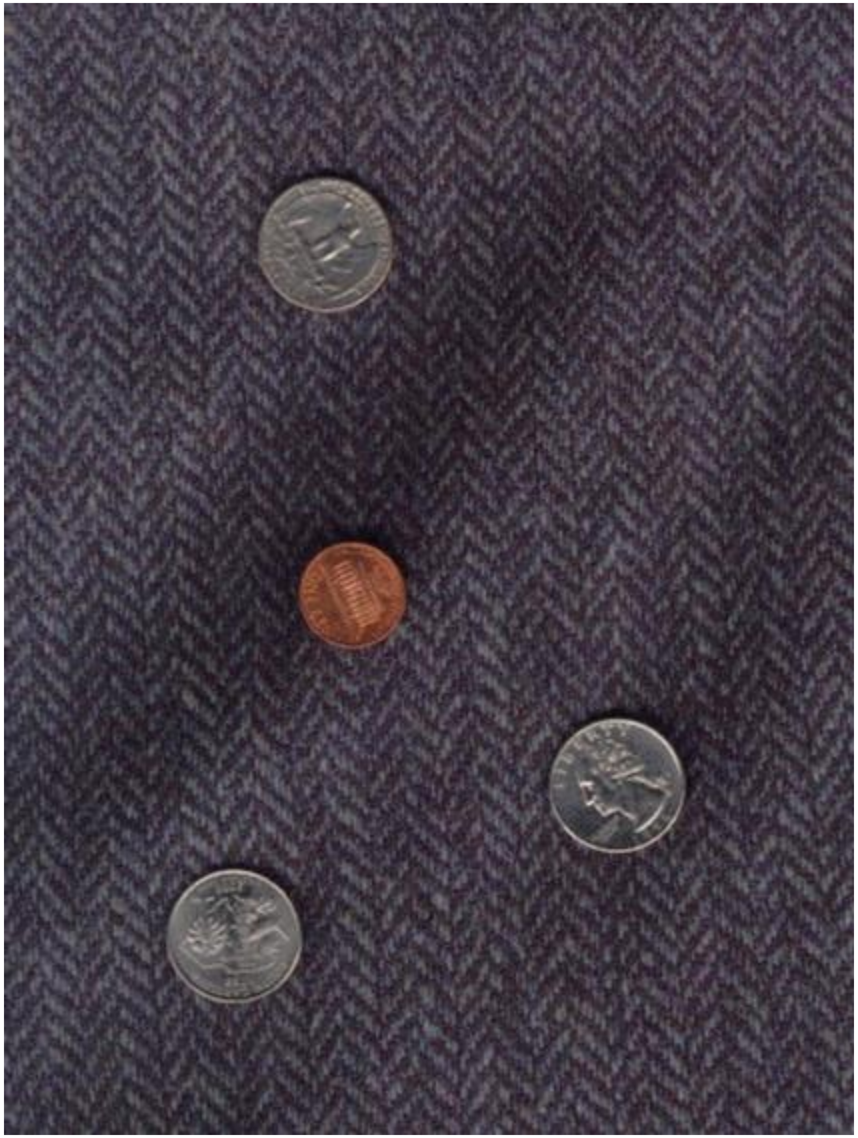
Application

Crosshair indicates results of Hough transform

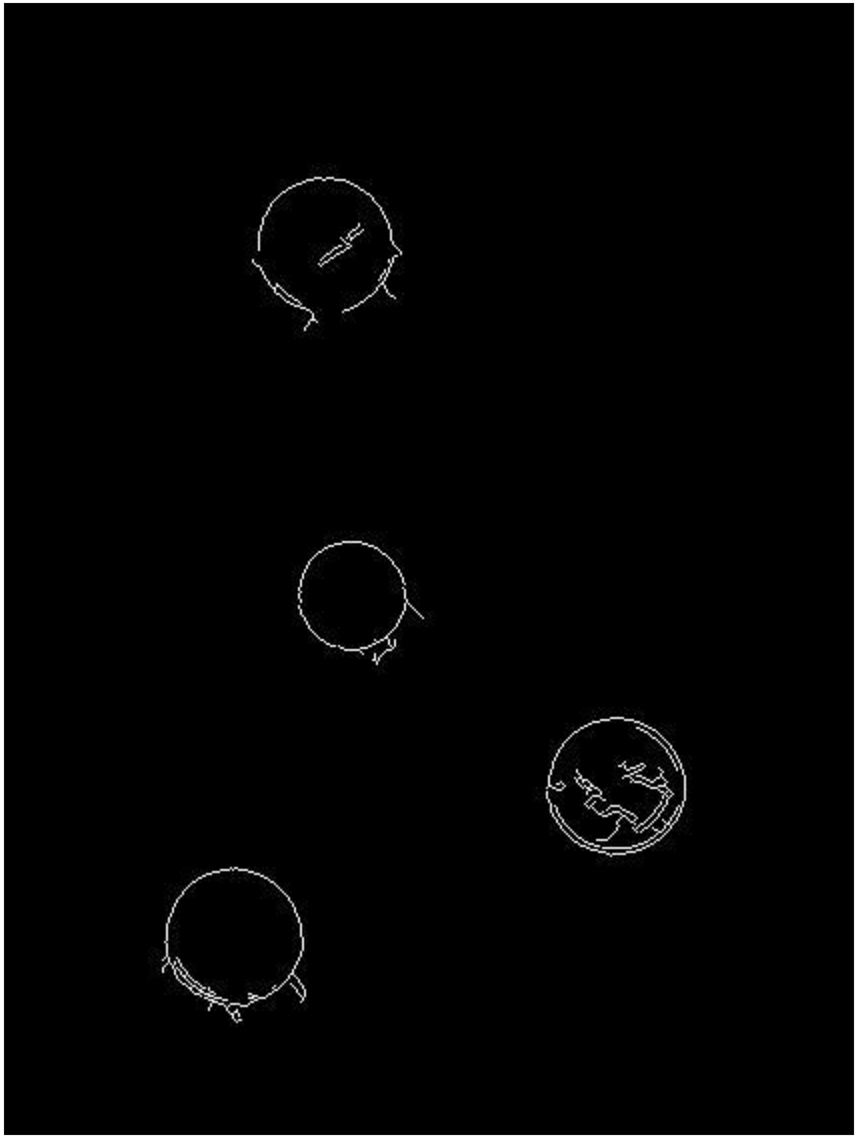
How ?
- using gradient
- using ?

Finding Coins

Original

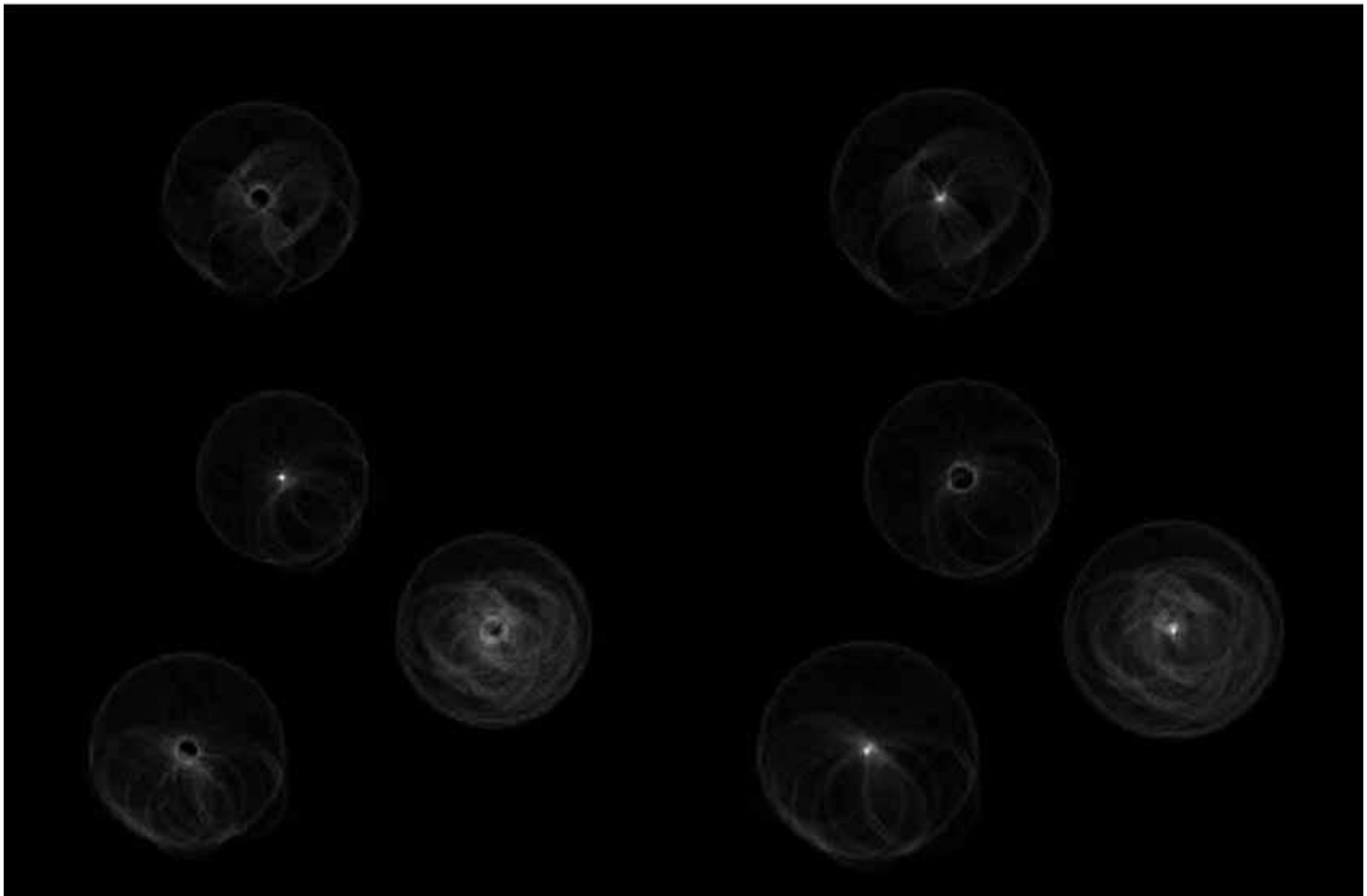


Edges (note noise)



Finding Coins (Continued)

For small radius For larger radius



Finding Coins (Continued)

Coin finding sample images from: Vivek Kwatra

Occlusions